NON-WORST-CASE RESPONSE-TIME ANALYSIS FOR REAL-TIME SYSTEMS DESIGN

A Thesis
Presented to
The Academic Faculty

by

Zhenwu Shi

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2014

NON-WORST-CASE RESPONSE-TIME ANALYSIS FOR REAL-TIME SYSTEMS DESIGN

Approved by:

Professor Fumin Zhang, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Yue Wang
Department of Mechanical Engineering
Clemson University

Professor Yorai Wardi
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Patricio Antonio Vela
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor A P Meliopoulos
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: March 6th, 2014

To my family ,

Thanks for your support and love.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere and greatest appreciation to people who have helped and supported me during my Ph.D study.

Foremost, I want to express my deepest gratitude to my advisor, Prof. Fumin Zhang, for his support, patience, guidance, and immense knowledge. He provided me with an excellent atmosphere for doing research, patiently corrected my writing, and financially supported my research. I would never have been able to finish my dissertation without his guidance and support.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Meliopoulos, Prof. Wardi, Prof. Wang, and Prof. Vela for their encouragement, insightful comments, and valuable suggestions. Special thanks goes to Prof. Meliopoulos and his student Evangelos Polymeneas for their discussions and suggestions on Chapter 5.

I would like to thank my fellow lab members and my friends who provided great collaboration and assistance during my study.

I would like to thank my parents Yuru Zhu and Xuping Shi, for giving birth to me at the first place and supporting me spiritually throughout my life.

Last but not the least, I would like to thank my wife, Judy Yan. She was always there cheering me up and stood by me through the good times and bad.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Real-time systems have been more and more widely observed in our daily life applications, such as network routers, industry robots, medical equipments, and automotive. In a real-time system, the correctness of operations depends not only on the logical results, but also on the time at which these results are available. A fundamental problem in designing real-time systems is to analyze response time of operations, which is defined as the time elapsed from the moment when the operation is requested to the moment when the operation is completed. Response time analysis is challenging due to the complex dynamics among operations. A common technique is to study response time under worst-case scenario. However, using worst-case response time may lead to the conservative real-time system designs. To improve the real-time system design, we analyze the non-worst-case response time of operations and apply these results in the design process.

The main contribution of this thesis includes mathematical modeling of real-time systems, calculation of non-worst-case response time, and improved real-time system design. We perform analysis and design on three common types of real-time systems as the real-time computing system, real-time communication network, and real-time energy management. For the real-time computing systems, our non-worst-response time analysis leads a necessary and sufficient online schedulability test and a measure of robustness of real-time systems. For the real-time communication network, our non-worst-response time analysis improves the accuracy of the process model for the model predictive control design based on the real-time communication network. For the real-time energy management, we use the non-worst-case response time to check whether the micro-grid can operate independently from the main grid.

CHAPTER I

INTRODUCTION

Real-time systems are playing an important role in our daily life. Examples of applications include network routers, industry robots, medical equipments, and automotive. The most distinguished feature of real-time systems is that the correctness of operations in real-time systems depends not only on logical results of operations, but also on the time when these results are produced. In other words, real-time systems have timing requirements that must be guaranteed. Such requirements can be evaluated by schedulability tests that check whether each operation can be completed before their deadlines.

An important technique in real-time systems design is to analyze response time of operations. Response time of an operation is defined as the time elapsed from the moment when the operation is requested to the moment when the operation is completed. Knowing response time of operations is necessary for 1) design of control applications on real-time systems, in which response time is a critical parameter in the design process, and 2) schedulability test of real-time systems, in which the schedulability of an operation is checked by comparing its response time with its deadline. Analyzing response time is a challenging task because of the complex relationship between operations and unpredictable delays. To simplify the analysis, most of existing work focuses on studying response time of operations under worst-case scenarios. However, analyzing worst-case response time for real-time systems design has its limitations. For example, a schedulability test that uses worst-case response time is pessimistic in the sense that it may fail a set of schedulable operations. Also, designing control applications based on the worst-case response time may be overly

conservative and lead to degraded control performance. Thus, the objective of our research is to develop non-worst-case response-time analysis and use this analysis technique to improve the design of real-time systems.

We deal with three broad classes of real-time systems, namely, real-time computing systems, real-time communication networks and real-time energy management. In real-time computing systems, each operation corresponds to computing a task on the processor. Analyzing non-worst-case response time of tasks leads to (1) a necessary and sufficient schedulability test that can be performed at runtime; and (2) robustness measure of real-time computing systems. For real-time communication networks, we study the design of a distributed control system based on a real-time communication network called the controller area network (CAN bus). Each operation is viewed as a control process that starts with sampling and ends with actuation. Knowing response time of control processes gives guidance on designing feedback control loops with desirable performance. Real-time energy management is a new topic that emerges in recent years. We study the real-time energy management in a micro-grid with renewable energy sources. Each operation can be viewed as execution of an electrical load. The on-site electricity generation in a micro-grid will fluctuate over time since the renewable energy is highly variable. Under the fluctuating power supply, we analyze response time of electrical loads and check whether micro-grids can guarantee the schedulability of electrical loads.

For each class of real-time system studied in this dissertation, non-worst-case response time of operations is studied through a unified procedure as follows.

1. As the first step, we establish mathematical expression for operations in each class of real-time systems by considering operation characteristics and constraints. More specifically, operations in the real-time computing systems are modeled as preemptive tasks with recurring instances; operations in the real-time communication network are modeled as non-preemptive message chains

that link each pair of sensor message and control message in a strict order; and operations in real-time energy management is modeled as electrical loads with recurring request and whether the operations is preemptive depends on the specific characteristics of electrical loads.

2. Based on the mathematical expression for operations, we develop analytical models that describe the scheduled behavior of operations in various real-time systems. In the real-time computing system, we first derive simple analytical models within a small sub-interval and then concatenate simple models to formulate a more complex analytical model within any large time interval. In the real-time communication network, we derive the analytical model as hybrid automata that represent both the continuous and discrete dynamics of operations. In the real-time energy management, the analytical model is a little bit involved as the operations have more various characteristics.

3. Finally, we derive non-worst-case response time from the analytical models and use them to improve the design for each class of real-time systems. More specifically, for the real-time computing system, we use the non-worst-case response time to derive an necessary and sufficient on-line schedulability test and a measure of robustness; for the real-time computing system, we use the non-worst-case response to obtain an accurate and reliable process model for the model predictive control design, which improves the control performance; and for the real-time energy systems, we use the non-worst-case response time to analyze the feasibility of the micro-grids.

The remaining part of this dissertation is presented as follows. Section 2 presents the literature review and background information of different classes of real-time systems. Section 3 analyzed the non-worst-case response time of tasks in a real-time computing system. Based on this result, we derive a necessary and sufficient

schedulaiblity test, and a method of measuring robustness. Section 4 designs model predictive controllers over the CAN bus, under the constraint of the time-varying delay induced by the real-time communication protocol. A hybrid timing model for the real-time scheduling of messages on the CAN bus is established, so that accurate estimation of response-time can be obtained at run time. Section 5 studies the operation of electrical loads in a micro-grid with renewable energy source. A feasibility test is established to check if the micro-grid can provide enough power to support the operation of electrical loads. Finally, Chapter 6 provides concluding remarks and the contributions of the thesis.

CHAPTER II

BACKGROUND

This chapter provides the background information of different types of real-time systems. In particular, the first part gives an overview of real-time computing systems. The second part presents the state of art techniques and applications of real-time communication networks. The third part discusses the latest progress in real-time energy management.

2.1 Real-time Computing Systems

Real-time systems are first developed for computing systems. In real-time computing systems, multiple tasks are computed simultaneously on the processor. A successful real-time computing system requires that all tasks can be computed before their respective deadlines. Since the processor can only compute one task at a time, a proper real-time scheduling is required to determine the order of task execution so that each task can meet its deadline.

Historically, the order of task execution in real-time computing systems was designed by static scheduling. In static scheduling, the schedule of tasks was constructed in an *ad hoc* manner off-line, based on the prior knowledge of task parameters, timing requirements, and scheduling constraints [89]. Once a static schedule is made, the tasks are executed at runtime according to that schedule. The static scheduling was widely used in the 1960s. However, during the 1970s and 1980s, it was understood that static scheduling can be very inflexible and difficult to maintain because the schedule produced off-line cannot be modified online [74]. This understanding leads to an explosion of research and publications on dynamic scheduling. In dynamic

scheduling, the order of task execution is decided at run time by continuously selecting one task from a set of tasks, according to scheduling algorithms. Dynamic scheduling is flexible and adaptive because the schedule is constructed at runtime. The implementation of dynamic scheduling requires a schedulability test to determine whether each task in a given set of tasks can be computed before their deadlines.

2.1.1 Schedulability Test

Research into schedulability test can be traced back to 1973, when Liu and Layland published a seminal paper [49] that is generally regarded as the foundational work in real-time computing systems. Liu and Layland [49] considered a real-time computing system with the following assumptions: (A1) all tasks are periodic, fully preemptible, and synchronized; (A2) all tasks have their relative deadlines equal to their periods; and (A3) system overhead and context switch can be ignored. Liu and Layland [49] introduced an idea of critical time instant, where all tasks are requested simultaneously. At the critical time instant, a task will endure the longest response time. Liu and Layland introduced timing analysis into the study of real-time scheduling. They proved that a set of tasks on the processor is schedulable under the rate monotonic scheduling (RMS) algorithm if their total processor utilization satisfies $\sum_{n=1}^{N} U_n \leq N(2^{1/N} - 1)$, in which U_n represents the processor utilization of an individual task τ_n, and N is the number of total tasks on the processor. Also, they proved that a set of tasks is schedulable under the earliest deadline first scheduling (EDF) algorithm if their total processor utilization satisfies $\sum_{n=1}^{N} U_n \leq 1$. Based on the similar timing analysis in [49] , extensive research has been conducted to improve the results made in Liu and Layland's work. Lehoczky et al [46] showed that the average processor utilization, for a large set of randomly chosen tasks schedulable under RMS, is approximately 88%. Tasks with offsets are studied in [66, 81]. Tasks with arbitrary deadlines are analyzed in [47] [83]. Abdelzaher et al [2] relaxed the

periodic restriction on tasks and derived an utilization bound for non-periodic tasks. Some papers [33, 47, 70, 75] considered tasks with shared resources and precedence constraints. Buttazzo et al [18] studied the overload situation, in which actual loads of tasks may exceed nominal estimations of loads. The timing analysis in Liu and Layland's work [49] focuses on the state of scheduled tasks at critical time instant. However, the state of scheduled tasks will not always stay at the critical time instant and will change as time propagates. In our research, we have introduced a new set of variables called the state vectors, which can represent the complete status of scheduled tasks at any time instant [96]. By introducing the state vectors, we can perform timing analysis beyond the critical time instant. Also, we have established an analytical model that describes the continuous evolution of the state vector as time propagates. Based on this timing model, we can have the accurate timing information of the scheduled task within any finite time window, which the timing analysis in Liu and Layland's work [49] does not provide. We will show in Chapter 3 knowing this accurate timing information can improve the schedulability test of the real-time computing systems.

Research progress has been made to extend scheduling from tasks with deterministic timing to sporadic tasks. Sporadic tasks are a special type of non-periodic tasks [60]. In sporadic tasks, the instances of tasks arrives at random time, but with a minimum inter-arrival interval [12]. Each sporadic task can be characterized by constant execution time, constant deadline, and constant period equal to the minimum interval-arrival. Based on this characterization, the timing analysis and schedulability test for periodic tasks can be easily extended to sporadic tasks. The timing analysis of sporadic tasks in a multi-criticality system is studied in [11, 68]. This timing analysis is based on the idea of critical time instant that was originally developed for periodic tasks. A online computation of the priority assignment for sporadic tasks is discussed in [32].

2.1.2 Response Time Analysis of Computing Systems

Using response time for schedulability test were introduced by Joseph and Pandya in 1986 [40] and Audsley et al in 1991 [8]. They propose an algorithm for calculating the worst-case response time of periodic tasks. The response-time analysis is in worst case because of the following reasons: (1) it uses upper bounds on execution times of tasks, commonly called worst-case execution times, for the analysis; and (2) it studies tasks at the critical time instant and hence calculates the longest time between the arrival of a task instance and its subsequent completion. Once the worst-case response time is given, schedulability of tasks can be checked by comparing their worst-case response time to their deadlines. The original algorithm has high computational complexity because it is realized through repeated iterations. To improve computational efficiency, many approaches have been proposed. Some papers [15] [76] studied how to use initial values at the beginning of the iterative procedure, and their simulations showed that the effective initial values can lead to fewer number of iterations. Fisher and Baruah [27] derived an approximation algorithm for response time analysis that has polynomial complexity. This algorithm is improved by an approach that partitions tasks into two subsets at each iteration [52]. One set of tasks was allocated CPU resources according to their utilization, while the other set of tasks use the remaining time of the processor. The response time of tasks in each set is analyzed separately. This approach can reduce up to 50.7% of the number of iterations of the original algorithm. At the same time, many research has been conducted to analyze the execution time of tasks. The static methods and measurement-based methods of analyzing the worst case execution time is summarized in [87]. Abdallah et al [1] introduced design of experiment (DOE) method into measuring the worst-case execution time. Relying on principles developed in DOE, the paper observes outputs and controls the inputs in a way to avoid measuring a response to every combination of inputs. Lu et al [53] proposed a statistical approach to analyze worst-case response

time analysis for system models with intricate task execution dependencies. Henia et al proposed a SymTA\S approach to analyze the worst-case response time in a complex real-time system with multiple components [36]. In SymTA\S approach, the time analysis of the global system is performed through compositional approaches in two phases. In the first phase, local scheduling analysis is performed for each component. In the second phase, the local analysis of different components is combined together. Through the two phases, SymTA\S will give out the worst-case response time analysis of the global system.

2.1.3 Application of Schedulability test

Schedulability tests are not only used in design process for verifying feasibility of tasks, but also used at runtime for admission control. The purpose of admission control is to accept or reject new tasks that are requested during the runtime of systems. A successful admission control must guarantee that the acceptance of a new task will not violate the schedulability of all existing tasks. Utilization bound-based schedulability test is widely used in admission control because it can run in constant time [3]. The utilization bound derived in Liu and Layland's work [49] can be used for the admission control of periodic taks under RMS and EDF scheduling. However, utilization-based admission control is pessimistic because it is only an sufficient admission test. Recent progress has been made in improving the performance of admission control. Bini et al [13] derived a hyperbolic bound for tasks scheduled under RMS. The hyperbolic bound is less pessimistic than Liu and Layland's utilization bound, and admission control that based on the hyperbolic bound can run in polynomial time. Andersson et al [5] proposed an exact admission control for periodic and aperiodic tasks under EDF scheduling. Zhang et al [95] proposed a fast schedulability analysis which is necessary and sufficient for EDF scheduling with arbitrary relative deadlines. In our previous work, we propose a dynamic schedulability test [96]. Our dynamic schedulaiblity

test provides a necessary and sufficent admission control for tasks scheduled under priority-based scheduling algorithms, including both EDF and RMS.

In recent years, a number of results have been reported on the co-design of scheduling and control. The idea of co-design is first introduced in [7]. It shows that the timings in real-time scheduling will affect the control performance. Zhang et al further explores the relationship between real-time task periods and the control performance of physical plants [97]. They define an operation point as a collection of periods of all real-time tasks. The goal is to find an optimal operation point that maximizes control performance under the schedulability constraint. The similar idea is extended to control design on automotive ECUs [30]. Since the automotive ECUs only support a finite number of task periods, the paper focuses on finding a possible sequence of task periods such that the resulting average task period is close to optimal ones. Some papers [56] proposed a control design strategy that can effectively compensate for delays introduced in real-time scheduling, provided that the values of delay is known. However, in order to fully exploit the benefit of integrating control design and real-time scheduling, many research issues are still open. As discussed in [88], one of the fundamental research issues is to establish an analytical timing model of real-time scheduling for control purpose, which may lead to better integration of control theory and real-time scheduling.

2.2 Real-time Communication Networks

Real-time systems have been developed for distributed environments. In distributed environments, each operation is realized by a set of distributed nodes exchanging information over some form of communication networks. To meet deadlines of operations, we require the communication among distributed nodes to be real-time. However, many general-purpose communication networks are not suitable for real-time communication. For example, Ethernet, as the most popular networking technology,

cannot satisfy the real-time requirement because of its inherent unpredictability. We study a specially designed real-time communication network called the controller area network (CAN).

2.2.1 CAN Protocol and Implementation

The CAN protocol is a communication protocol aimed for providing real time properties over a wired network. The CAN protocol covers only the Data Link Layer and some abstract requirements of the Physical Layer of the OSI reference model. The remaining portion of the OSI Reference Model was purposely left out of the CAN protocol to enable system designers to adapt and optimize the CAN protocol for maximum flexibility. We give a brief overview of the CAN protocol and the CAN nodes. More detailed information can be found in the technical manuals available online [28].

2.2.1.1 Message Frame

The CAN protocol defines a standard data message that encapsulates information transmitted between a source node and one or more receivers. As shown in Figure 1, the data message is composed of seven fields: an SOF field, which represents the start of the message; an identifier field, which is a unique number assigned to the message; a control field, which indicates the length of the data field; a data field, which contains the information encapsulated in the message; a CRC field, which checks the integrity of the message; an ACK field, which acknowledges the reception of the message; and an EOF field, which represents the end of the message.

Figure 1: A standard message frame in CAN

The CAN protocol is a content-based protocol rather than an address-based protocol such as TCP [67]. Unlike the latter, which assigns each message an explicit destination address, the CAN protocol assigns each message a unique identifier. When a node attempts to transmit a message, it broadcasts the message on the CAN bus, and all nodes receive the message from the CAN bus. By checking the identifier, receivers decide whether or not the message is intended for them. Such content-based protocol has two major advantages. First, a message can be destined for any number of nodes simultaneously, which increases the utilization of the CAN bus. Second, additional nodes can be added to the existing CAN bus without the necessity to reprogram all other nodes to recognize this addition, which increases the flexibility of the CAN bus. However, content-based protocol requires bit synchronization among all nodes so that each node will have the same interpretation of the received message. This bit synchronization can be easily realized in the wired communication, but is difficult for the wireless communication due to the multi-path effect.

2.2.1.2 Bus Arbitration

The broadcast communication scheme of the CAN bus implies that only one node can transmit a message at a time. If two or more nodes attempt to transmit messages at the same time, collisions will happen on the CAN bus. To resolve such collisions, CAN uses an arbitration scheme known as the carrier sense multiple access with bitwise arbitration (CSMA/BA) [28]. CSMA/BA interprets the identifier of each message as its priority: the smaller the value of the identifier field is, the higher the priority of the message is. Since the identifier is unique, each message has a unique priority. Therefore, whenever two or more nodes attempt to transmit messages at the same time, the node transmitting the highest priority message will be granted final access to CAN, and the other nodes will have to defer their message transmission until CAN becomes idle.

Using identifiers as priorities is enabled by the wired-AND physical property of the CAN bus: if multiple nodes are transmitting messages at the same time and one of the nodes transmits a logic bit "0", the value of the CAN bus will be "0"; and only if all of the nodes transmit a logic bit "1" will the value on the CAN bus be "1". Based on this physical property, CSMA/BA performs arbitration as follows: (1) the arbitration starts from the first bit in the identifier field and ends at the last bit in the identifier field; (2) each node transmits the identifier of a message while monitoring the value on the CAN bus; and (3) if a node's transmitted bit differs from the value on the CAN bus, the node detects a collision and aborts its message transmission; if a node's transmitted bit is identical to the value on the CAN bus, the node continues its message transmission. Since each message has a unique identifier, a node transmitting the last bit of the identifier without detecting a collision must be transmitting the highest priority message, and can continue transmitting the remaining part of the message. According to the above arbitration process, we can see that: (1) The logic bit "0" can always win arbitration over the logic bit "1", therefore the message with a lower value in the identifier field has a higher priority; (2) the highest-priority message wins arbitration without being disturbed since the transmission of lower priority messages will automatically back off and wait; and (3) using identifier of messages as their priorities makes CAN particularly suitable for real-time communication. The drawback is that there can only be a finite number of different types of messages.

2.2.1.3 Implementation of the CAN bus

Figure 2 illustrates the functional structure of a node on the CAN bus. The node consists of four modules: an application module, which represents a device communicating over the CAN bus, such as sensors, actuators and controllers; a host processor, where user-specified functions can be implemented to tailor the CAN protocol for

different applications; a CAN controller, which implements the CAN protocol of receiving and transmitting messages; and a CAN transceiver, which converts the data from the CAN controller to signals on the physical bus. Therefore, the application module views the other three modules as a CAN-interface, through which information can be exchanged with the CAN bus.

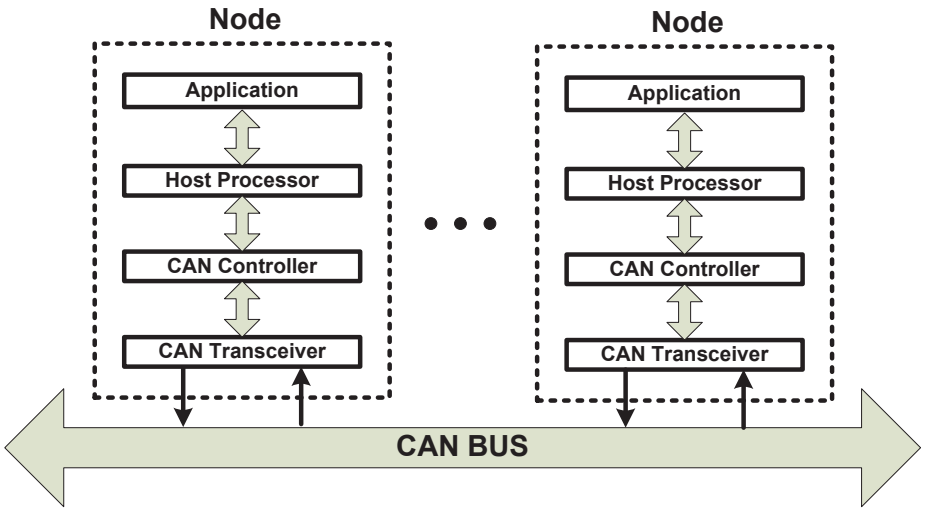


Figure 2: Implementation of the CAN bus

2.2.2 Response Time Analysis of the CAN bus

Response-time analysis is of fundamental importance to validate the real-time performance of CAN. In 1994, Tindell et al proposed a first solution for analyzing the worst-case response time of CAN [80,82]. This solution was later recognized by Volvo Car Corporation and successfully used as the theoretical foundation for commercial CAN schedulability analysis tools [19]. However, the analysis in [80, 82] was later found to be flawd under high network loads. Davis and Bril [21] analyzed the cause of this flaw and provided a set of formulas for corrections. Another problem of Tindell's analysis is that it relies on some ideal assumptions of CAN that may not be supported in real applications. Many studies have attempted to relax these assumptions. Fault models are developed to analyze response time of CAN when the transmission errors happen [16,17,69]. Some papers [31,91] analyzed response time of

CAN when messages are not starting simultaneously. The effect of hardware limitations on the worst-case response time of CAN is studied in [22,43,44,63]. Probabilistic response-time analysis of CAN is recently reported in [9, 92].

2.2.3 CAN-based Control System

The CAN-based control system is a system in which feedback control loops are closed via the CAN bus. The integration of the CAN bus and feedback control loops can bring many benefits, such as system flexibility and easy maintenance. In recent years, a number of results have been reported on designing CAN-based Control System. The proper design of self-trigger controllers can significantly reduce the number of required messages, and hence preserve the communication bandwidth of CAN for other applications [6]. Marti et al [55] proposed an optimal strategy to allocate communication bandwidth to different control loops implemented on CAN. The effect of response time on the control performance is analyzed in [38].

Model predictive control (MPC) is gaining popularity in industry. MPC iteratively uses a process model of the physical system to predict and optimize future system behaviors at each time instant [71]. The implementation of MPC for safety-critical systems built on the CAN bus is an open challenge that has not been sufficiently addressed in the literature. Most of work in the literature focuses on the integration of MPC and general purpose network using communication protocol such as TCP and UDP [29,37,50,51,61]. However, the communication protocol of general purpose network is different from the real-time communication protocol on the CAN bus. Therefore, these work cannot be directly applied to the MPC design for the CAN bus. We will propose one methodology that focuses on handling the timing constraints caused by the CAN scheduling protocol in Chapter 4. To the best of our knowledge, this study does not exist in literatures reviewed.

Effective MPC designs rely on accurate, high fidelity process models. However,

real-time scheduling of messages on the CAN bus incurs time-varying delays into feedback control loops. Such time-varying delays make it difficult to derive an reliable process model for MPC design. Existing real-time scheduling analysis of the CAN bus focuses on modeling time-varying delays as either constant values in worst-case scenarios [21, 79, 82] or stochastic variables obeying certain distribution [93]. These results do not provide a process model with sufficient accuracy. Moreover, many control systems nowadays are operating in dynamic and uncertain environments. As a result, the system workload will change accordingly. For instance, some messages on the CAN bus may need to be removed in some cases, while new messages may be added in other cases. This variability of messages inevitably further increases delay variation in feedback control loop.

2.3 Real-time Energy Management for Micro-grids

Reliable operation of the power grid requires a balance between providers' supply and consumers' demand at all times. In traditional power grid, the consumers' demand request is served instantaneously and continuously. Therefore, providers must build enough reserve plants to guarantee supply for the maximum anticipated demand. However, this method is very inefficient as many reserve-power plants are only in use for a few hours each year [26]. Recently, the development of the smart grid increases the opportunity of energy management [58], which optimize the execution of consumers' demand in response to supply conditions [4] [73]. The success of the energy management is based on the fact that many electrical loads are deferrable and interruptible for a few minutes or hours at little or no cost. Therefore, energy management can selectively activate/deactivate deferrable loads in response to electricity supply conditions.

2.3.1 Real-time Energy Management

A number of works have been reported on applying real-time scheduling techniques for energy management. Facchinetti et al [24] proposed an approach to model electrical loads using real-time parameters. Based on the new model, the paper showed that the peak power can be reduced by partitioning electrical loads into groups and scheduling each group independently through the EDF algorithm. The similar approach is applied to model electrical vehicles (EV) charging as real-time computing tasks and develop scheme for real-time EV charging systems [41]. Xu et al [90] proposed a scheduling algorithm that gives higher priorities to vehicles with less laxity and longer remaining charging time. Simulations showed that this algorithm gave optimal economic benefits. Subramanian et al [77] investigates three different algorithms for scheduling deferrable electrical loads under varying energy supply. It proved that the EDF algorithm is an optimal scheduling algorithm if the power consumption of electrical loads is not constrained. Also, it showed that both the least laxity first and the receding horizon control algorithms can effectively coordinate the operation of electricity loads so that the aggregate power consumption well approximated the varying energy supply. However, these works simply model electrical loads with power and execution time, but not internal physical dynamics of electrical loads. In [25], electrical loads are represented by an dynamic model . The constraints on physical dynamics are translated into timing constraints on real-time scheduling.

2.3.2 Micro-grid Technology

Micro-grids are modern, small-scale versions of the centralized power grid [34] [45]. They achieve specific goals required by the local community being served, such as power reliability, carbon emission reduction, diversification of energy sources and cost reduction. Micro-grids are capable of operating in parallel with (grid connected mode), or independently from (island mode), the national grid. In the island mode,

the micro-grid is disconnected from the national grid, but is autonomously energized by on-site electricity generations. This autonomous feature makes micro-grids an ideal energy solution for isolated, small areas that may never be connected to the national grid due to their remoteness [72].

Like the centralized power grid, micro-grids generate, distribute and regulate the flow of electricity to consumers, but do so locally. Operation of micro-grids relies on three major components as on-site generation, energy storage, and control systems [42]. The on-site generation produces electricity from small sources of energy located at or near the point of use. Typical sources of energy in micro-grids include photovoltaic, wind, fuel cells and micro-turbines. The energy storage, such as batteries and super-capacitors, is particularly useful for micro-grids with intermittent renewable energy as the means of the on-site generation, where the generation cannot match the load demand at all times. The energy storage saves extra energy whenever the on-site generation exceeds the load demands, and provides power whenever the on-site generation is insufficient to supply the load demands. By decoupling the generation source from the load, energy storage enhances the stability and efficiency of micro-grids. The control system plays a central role in the micro-grid. Its main object is to continuously supply power to the loads despite the changes in the system. A micro-grid may be switch between grid-connected mode and island mode. One or more on-site generation units may connect to or disconnect from the micro-grid. The load demand might significantly change within a short time. Under all these circumstances, the control system shall ensure that power is supplied to the loads with acceptable voltage and frequency characteristics. Moreover, the control system can apply appropriate real-time management strategy to optimize the execution of electrical loads in response to the electricity generation.

Energy management is an important issue in micro-grids [62]. A lot of recent work has been reported on optimizing the operation schedule of energy management

under different cost functions [62] [10] [59]. A central control system for micro-grids is introduced in [85]. This control system optimizes the energy management during interconnected operation by adjusting the production of the on-site generation and power exchanges with the main power grid. Energy management for micro-grids in both commercial buildings and residential homes have been analyzed in [54] [94].

CHAPTER III

SCHEDULABILITY OF REAL-TIME COMPUTING SYSTEM

In real-time computing systems, multiple tasks will share the same computation resource (i.e. processor) for their operations. Each task is a software program consisting of a sequence of instructions that are repeatedly executed on the processor. Each activation of a task may have its own release time, execution time, and deadline. A successful real-time computing system must guarantee that each activation of a task can finish execution by its deadline.

3.1 Task Characteristics

We consider a set Γ of N independent hard real-time tasks running concurrently on a single processor, i.e. $\Gamma = \{\tau_1, \cdots, \tau_N\}$. Each task in Γ consists of an infinite sequence of instances. Let τ_n be any task in Γ. We use the notation $\tau_n[k]$ to represent the k-th instance of τ_n. The instance $\tau_n[k]$ is characterized by its release time $\alpha_n[k]$, execution time $C_n[k]$, priority $P_n[k]$, and relative deadline $T_n[k]$ measured from $\alpha_n[k]$. The absolute deadline of $\tau_n[k]$ is then defined as $\alpha_n[k] + T_n[k]$. For the purpose of being general, we assume all tasks in Γ are *acyclic* tasks rigorously defined as follows

Definition 3.1.1 *A task τ_n is* acyclic *if and only if it satisfies the following properties:*

1. *different instances of τ_n can have different execution times and different relative deadlines, as long as $0 \leq C_n[k] \leq T_n[k]$ and $T_n[k] > 0$ for all k;*

2. *the release time of a new task instance coincides with the absolute deadline of*

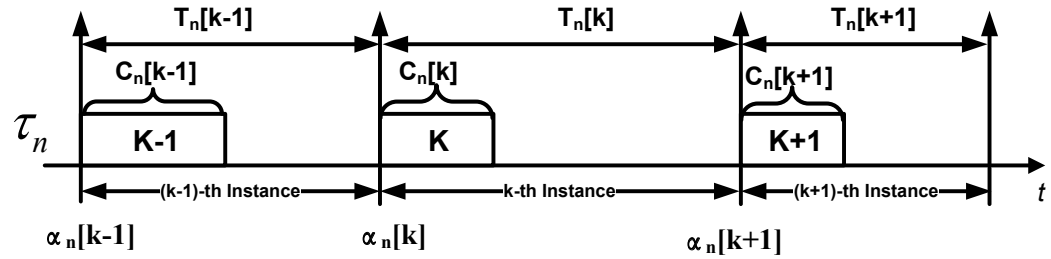


Figure 3: Illustration of one acyclic task scheduled on a processor. Three task instances indexed by $k-1$, k, and $k+1$ are plotted.

the previous task instance of the same task, i.e. $\alpha_n[k+1] = \alpha_n[k] + T_n[k]$ *for all* k.

Figure 3 shows an example of acyclic tasks. The horizontal line represents the progression of time. The upward arrows represent the release times of new task instances, the rectangles represent the execution time of task instances, the time intervals between any two consecutive arrows represents the relative deadlines. Note the priority of tasks is a little bit involved and we will give detailed discussions in Section 3.2.2. The reason we use the acyclic task model is that: (1) any periodic task can be represented by an equivalent acyclic task. For example, a periodic task with execution time 2 and period 5 can be represented by an acyclic task with $C_n[k] = 2$ and $T_n[k] = 5$ for all k; (2) any set of non-periodic tasks, i.e. tasks with irregular releasing instances, can be also represented by an equivalent set of acyclic tasks.

We want to model the scheduled behaviors of the real-time tasks at any time t. Therefore, we define the task characteristics in continuous time domain as follows

Definition 3.1.2 *At any time* t*, an instance of* τ_n *is effective if and only if it is the nearest instance released before or at time* t*, i.e.* $\tau_n[k]$ *is effective at time* t *if and only if*

$$\alpha_n[k] \leq t < \alpha_n[k+1]. \tag{1}$$

Definition 3.1.3 *At any time* t*,* $C_n(t)$*,* $T_n(t)$*, and* $P_n(t)$ *are respectively defined as*

the execution time, the period, and the priority of the effective instance of τ_n, i.e.

$$C_n(t) = C_n[k] \quad T_n(t) = T_n[k] \quad P_n(t) = P_n[k], \quad \text{if } \alpha_n[k] \leq t < \alpha_n[k+1]. \tag{2}$$

3.2 Dynamic Timing Model

In this section, we develop a mathematical model that describe how a set of tasks $\Gamma = \{\tau_1, \cdots, \tau_N\}$ are scheduled on the processor within $[t_a, t_b]$.

3.2.1 State Vector

State vectors are generally used in differential or difference equations that models dynamic systems behaviors [**?**]. To describe the scheduled behaviors of Γ at any time t, we define a state vector $Z(t) = [S(t), R(t), O(t)]$ as follows.

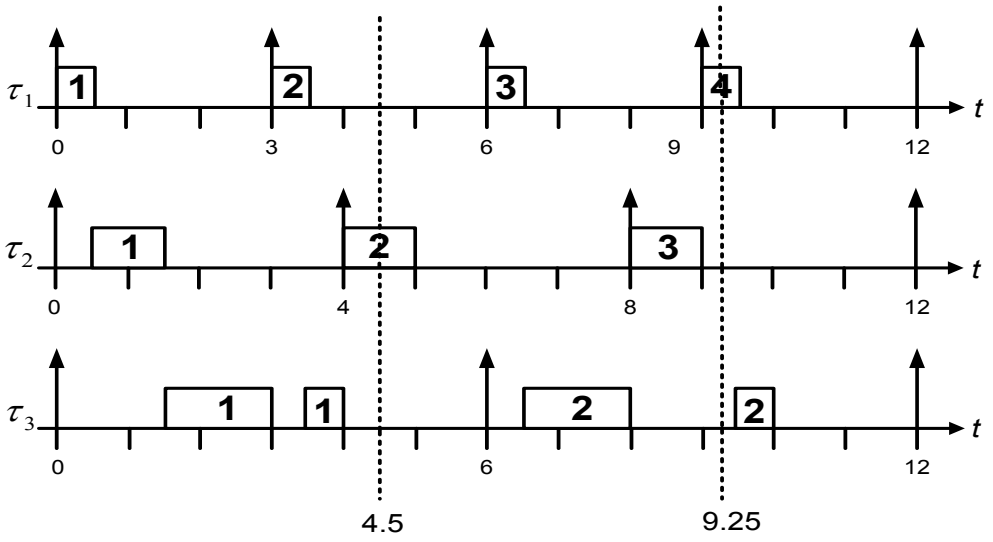
Definition 3.2.1 *The dynamic inter-release time is defined as $S(t) = [s_1(t), \cdots, s_N(t)]$, where $s_n(t)$, for $n = 1, 2, \cdots, N$, denotes how long after t the next instance of τ_n will be released.*

Definition 3.2.2 *The residue is defined as $R(t) = [r_1(t), ..., r_N(t)]$, where $r_n(t)$, for $n = 1, ..., N$, denotes the remaining execution time of the effective instance of τ_n* after *time t.*

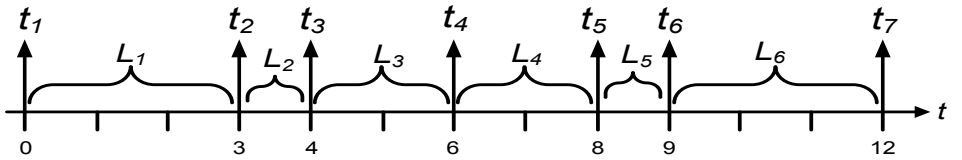
Definition 3.2.3 *The dynamic response time is defined as $O(t) = [o_1(t), ..., o_N(t)]$, where $o_n(t)$, for $n = 1, 2, ..., N$, denotes the length of time from the release of τ_n to time t or time when the effective instance of τ_n is completed, whichever occurs first.*

We use the following example to further explain the meaning of S, R and O. For ease of demonstration, we consider three periodic tasks.

Example 3.2.4 *Consider tasks $\{\tau_1, \tau_2, \tau_3\}$ with $[C_1(t), C_2(t), C_3(t)] = [0.5, 1, 2]$ and $[T_1(t), T_2(t), T_3(t)] = [3, 4, 6]$ for $t \in [0, +\infty)$. The three periodic tasks are scheduled under a fixed priority preemptive scheduling algorithm such that the priority of τ_1 is higher than τ_2, and the priority of τ_2 is higher than τ_3.*



(a) Scheduled Behavior



(b) Fixed Priority Window

Figure 4: Three acyclic tasks scheduled on one processor

Figure 4.(a) demonstrates the scheduled behavior of $\{\tau_1, \tau_2, \tau_3\}$ on one processor. We use the same plotting conventions as in Figure 3, where the upper arrows indicate the times of arrival of the task instances. It can be observed that the computation of lower priority tasks are interrupted by the computation of higher priority tasks. At time $t = 4.5$, $\tau_1[2]$, $\tau_2[2]$ and $\tau_3[1]$ are the effective instances of the three tasks. They arrive at 3, 4 and 0 respectively.

We can observe that at time $t = 4.5$, the next instance of $\tau_1[2]$, $\tau_2[2]$ and $\tau_3[1]$ will be released at 6, 8 and 6 respectively. Thus, according to Definition 3.2.1, the dynamic inter-release time are

$$S(4.5) = [s_1(4.5), s_2(4.5), s_3(4.5)] = [6 - 4.5, 8 - 4.5, 6 - 4.5] = [1.5, 3.5, 1.5]. \quad (3)$$

After $t = 4.5$ only τ_2^2 has not finished computing. Therefore, the remaining computing times after $t = 4.5$ are 0, 0.5 and 0. By Definition 4.3.2, we have

$$R(4.5) = [r_1(4.5), r_2(4.5), r_3(4.5)] = [0, 0.5, 0]. \tag{4}$$

For $\tau_1[2]$, it arrives at 3 and finishes computation at time 3.5. Therefore, the dynamic response time for $\tau_1[2]$ at time 4.5 is 0.5, i.e. $o_1(4.5) = 0.5$. For $\tau_2[2]$, it arrives at 4 and has NOT finished computation at time 4.5. Therefore, the dynamic response time for $\tau_2[2]$ at time 4.5 is 0.5, i.e. $o_2(4.5) = 0.5$. For $\tau_3[1]$, it arrives 0 and finishes computation at time 4. Therefore, the dynamic response time for $\tau_3[1]$ at time 4.5 is 4, i.e. $o_3(4.5) = 4$. Thus, according to Definition 3.2.3, we have that

$$O(4.5) = [o_1(4.5), o_2(4.5), o_3(4.5)] = [0.5, 0.5, 4]. \tag{5}$$

Similarly, at $t = 9.25$, we have the state vector as

$$S(9.25) = [2.75, 2.75, 2.75], \quad R(9.25) = [0.25, 0, 0.5], \quad O(9.25) = [0.25, 0.5, 2]. \tag{6}$$

It is worth mentioning that the dynamic-response time $o_n(t)$ records the delay of τ_n since its time of release. The value of $o_n(t)$ is reset to zero whenever an new instance of τ_n is released, continuously increase until the instance of τ_n finishes computation, and keep constant afterwards.

3.2.2 Scheduling Algorithm

Scheduling algorithms are disciplines used for distributing resources among tasks with simultaneously and asynchronously requests. It decides which of the outstanding requests is to be allocated resources. In real-time computing systems, scheduling algorithms will assign tasks different priorities and the allocation of resources will follow task priories in decreasing order.

In this section, we will now rigorously define scheduling algorithms, which will be used by our mathematical models for the scheduled tasks later. Let $\mathcal{A} = \{1, 2, ..., N\}$

be the set of indices of tasks, $\mathcal{R}^+$ be the positive real numbers, and $\mathcal{Z}^+$ be the positive integers. One way to formally define a scheduling algorithm is to establish the mapping between the scheduling algorithm and task priorities as follows.

Definition 3.2.5 *A* scheduling algorithm *is a set-valued map between* $\mathcal{A} \times \mathcal{R}^+$ *and* $\mathcal{Z}^+$. *It is parameterized as* $P_n(t)$ *where* $n \in \mathcal{A}$ *and* $t \in \mathcal{R}^+$ *so that* $P_n(t) < P_m(t)$ *if the task* τ_n *is assigned the higher priority than* τ_m .

For example, assume all tasks are periodic and the RMS algorithm [49] is used to assign fixed priorities. Suppose that tasks are labeled according to the length of their periods i.e. tasks with longer periods have larger indices. Then we have:

$$P_n(t) = n. \tag{7}$$

Consider another example where a dynamic priority scheduling algorithm such as the EDF algorithm is used. Then, the values of $P_n(t)$ depend on $S(t)$. At any time t, the EDF assigns higher priorities to the tasks whose effective instances have closer absolute deadlines. According to the definition of $S(t)$, tasks whose effective instances having closer absolute deadlines also have smaller dynamic deadlines. Thus, for the EDF, the tasks with smaller values of $s_n(t)$ are assigned higher priorities. When two tasks have the same dynamic deadlines, we assume that a higher priority is assigned to the task with a smaller index. Hence, the task priority $P_n(t)$ can be expressed as

$$P_n(t) = \text{Card}\left(\{\, i \mid 1 \le i \le N, s_i(t) < s_n(t) \,\}\right). \tag{8}$$

where the function Card$(\cdot)$ measure the number of elements in a set.

3.2.3 Sub-interval Window

For the ease of derivation, we further divide $[t_a, t_b]$ into a series of consecutive sub-intervals denoted as $[t_w, t_{w+1}^-]$. Note t_{w+1}^- to denote the time point that is less than t_w but is arbitrarily close to t_w. Thus, the sub-interval $[t_w, t_{w+1}^-]$ is equivalent to

$[t_w, t_{w+1})$. We define each sub-interval as a finite time window satisfy the following conditions

Definition 3.2.6 *Each sub-interval* $[t_w, t_{w+1}^-]$ *is defined as an finite time window such that task instances only arrive at* t_w*, but not at any other time point within* $[t_w, t_{w+1}^-]$*.*

In other words, task instance can only arrive at either t_w or t_{w+1}, but not within the sub-interval.

To better understand this definition, we consider an example of three acyclic tasks in Figure 4.(b). In this example, a large time interval $[0, 12]$ is divided into a series of consecutive sub-intervals satisfying the conditions in Definition 3.2.6. The advantage of such division is that the evolution of $Z(t)$ within each sub-interval $[t_w, t_{w+1}^-]$ is relatively easier to model. Then, the models in individual sub-intervals can be concatenated together to constitute the more complex model for the evolution of $Z(t)$ within $[t_a, t_b]$.

Next, we study how to divide $[t_a, t_b]$ into consecutive sub-intervals. We denote the length of each sub-interval as L_w, i.e

$$L_w = t_{w+1} - t_w, \tag{9}$$

then each sub-interval $[t_w, t_{w+1}^-]$ can be rewritten as $[t_w, (t_w + L_w)^-]$. Hence, the partition of $[t_a, t_b]$ into sub-intervals is determined by the window length L_w for $w = 1, 2, \cdots$. To determine the value of each L_w, we have the following claim

Claim 3.2.7 *For a set of acyclic tasks, at the beginning of any sub-interval, i.e.* t_w*, if we choose* $L_w \leq \min\{s_1(t_w), ..., s_N(t_w)\}$*, then* $[t_w, (t_w + L_w)^-]$ *is a sub-interval satisfying Definition 3.2.6; otherwise,* $[t_w, (t_w + L_w)^-]$ *dose not satisfy Definition 3.2.6.*

Proof 3.2.8 *At the beginning of any sub-interval, i.e.* t_w*, consider the dynamic release-time* $S(t_w) = [s_1(t_w), ..., s_N(t_w)]$*, as defined in Definition 3.2.1. According to*

the definition of $S(t_w)$, we know that the next task instance after t_w will be released at $t_w + \min\{s_1(t_w), ..., s_N(t_w)\}$.

If we choose $L_w = \min\{s_1(t_w), ..., s_N(t_w)\}$, then no new instance of any task is released between $(t_w, t_w + L_w)$. Therefore, $[t_w, (t_w + L_w)^-]$ is a sub-interval satisfying Definition 3.2.6.

On the other hand, if we choose $L_w > \min\{s_1(t_w), ..., s_N(t_w)\}$, the next task instance after t_w will be released between $(t_w, t_w + L_w)$. Therefore, $[t_w, (t_w + L_w)^-]$ is not a sub-interval satisfying Definition 3.2.6.

The division of $[t_a, t_b]$ into consecutive sub-intervals is carried out using the following procedure. At the beginning of the first sub-interval, let $t_1 = t_a$, we choose the first window length L_1 and the end of the sub-interval $t_2 = t_1 + L_1$ to make $[t_1, t_2^-]$ a sub-interval. Then by choosing the window length L_2 and letting $t_3 = t_2 + L_2$, the second sub-interval $[t_2, t_3^-]$ can be made a fixed priority window. The process is repeated until one sub-interval reaches the ending time t_b. According to Claim 3.2.7, we know that the largest possible window length L_w can be expressed as

$$L_w = \min\{s_1(t_w), ..., s_N(t_w), t_b - t_w\} \tag{10}$$

where the extra term $t_b - t_w$ guarantees that the division procedure stops at time t_b. A larger window length is preferred since it reduces the complexity in modeling the behaviors of tasks. Based on Equation (10), we know that the end of the current sub-interval is

$$t_{w+1} = t_w + L_w = t_w + \min\{s_1(t_w), ..., s_N(t_w), t_b - t_w\} \tag{11}$$

3.2.4 Evolution of State Vector

The evolution of $Z(t)$ within any sub-interval $[t_w, t_{w+1}^-]$ can be derived through two steps: from t_w^- to t_w, and then from t_w to any time $t_w + \epsilon \in [t_w, t_{w+1}^-]$.

From t_w^- to t_w: we first discuss the evolution of $[s_n(t), r_n(t), o_n(t)] \in Z(t)$ from t_w^- to t_w. For any task τ_n, the values of the state vector at time t_w, i.e. $[s_n(t_w), r_n(t_w), o_n(t_w)]$, depend on whether an new instance of τ_n is released at t_w.

(1) if no instance of τ_n is released at t_w, we have that $s_n(t_w^-) > 0$. In this case, the state vector $[s_n(t), r_n(t), o_n(t)]$ holds their values from t_w^- to t_w, i.e.

$$\text{if } s_n(t_w^-) > 0: \; s_n(t_w) = s_n(t_w^-) \quad r_n(t_w) = r_n(t_w^-) \quad o_n(t_w) = o_n(t_w^-) \tag{12}$$

(2) if an instance of τ_n is released at t_w and the old instance of τ_n has finished computation, we have that $s_n(t_w^-) = 0$ and $r_n(t_w^-) = 0$. In this case, the state vector $[s_n(t), r_n(t), o_n(t)]$ is updated according to the characteristics of newly released instance, i.e.

$$\text{if } s_n(t_w^-) = 0 \text{ and } r_n(t_w^-) = 0: \; s_n(t_w) = T_n(t_w) \quad r_n(t_w) = C_n(t_w) \quad o_n(t_w) = 0 \tag{13}$$

(3) if an instance of τ_n is released at t_w and the old instance of τ_n has NOT finished computation, we have that $s_n(t_w^-) = 0$ and $r_n(t_w^-) > 0$. In this case, an overload situation happens. To handle the overload situation, we use a simple yet efficient method of skipping any new instance of τ_n until the old instance of τ_n has finished its computation. Thus, we have that

$$\text{if } s_n(t_w^-) = 0 \text{ and } r_n(t_w^-) > 0: \; s_n(t_w) = T_n(t_w) \quad r_n(t_w) = r_n(t_w^-) \quad o_n(t_w) = o_n(t_w^-) \tag{14}$$

According to Equation (12), (13), and (14), we can express the evolution of the state vector $[s_n(t), r_n(t), o_n(t)]$ from t_w^- to t_w as

$$\begin{aligned}
s_n(t_w) &= s_n(t_w^-) + (1 - \text{sgn}(s_n(t_w^-)))\, T_n(t_w) \\
r_n(t_w) &= \text{sgn}(s_n(t_w^-) + r_n(t_w^-))\, r_n(t_w^-) + (1 - \text{sgn}(r_n(t_w^-)))\, (1 - \text{sgn}(s_n(t_w^-)))\, C_n(t_w) \\
o_n(t_w) &= o_n(t_w^-)\, \text{sgn}(s_n(t_w^-)) + o_n(t_w^-)\, \text{sgn}(r_n(t_w^-))(1 - \text{sgn}(s_n(t_w^-))),
\end{aligned} \tag{15}$$

in which sgn is a signum function, such that $\text{sgn}(x) = 1$ if $x > 0$, $\text{sgn}(x) = 0$ if $x = 0$, and $\text{sgn}(x) = -1$ if $x < 0$.

From t_w to $t_w + \epsilon$: we then discuss the evolution of $[s_n(t), r_n(t), o_n(t)]$ from t_w to any time $t_w + \epsilon$, where $\epsilon \in [0, L_w)$. Hence, we have that $t_w + \epsilon \in [t_w, t_{w+1}^-]$.

(1) For the dynamic inter-release time $s_n(t)$, we know that the next instance of τ_n is released at time $(t_w + \epsilon) + s_n(t_w + \epsilon)$. Since this release-time can be also expressed as $t_w + s_n(t_w)$, we must have $s_n(t_w + \epsilon) + t_w + \epsilon = q_n(t_w) + t_w$. Therefore, the equation for $s_n(t_w + \epsilon)$ is written as

$$s_n(t_w + \epsilon) = t_w + s_n(t_w) - (t_w + \epsilon) = s_n(t_w) - \epsilon. \tag{16}$$

(2) For the residue $r_n(t)$, we know that the computation of τ_n is preempted until the computation of all higher priority tasks are completed. Then, the amount of time within $[t_w, t_w + \epsilon]$ that is available to compute τ_n is

$$\max\{\ 0,\ \epsilon - \sum_{P_i(t_w) < P_n(t_w)} r_i(t_w)\ \}. \tag{17}$$

where $\sum_{P_i(t_w) < P_n(t_w)} r_i(t_w)$ denotes the summation of residues of all tasks with higher priorities than τ_n at time t_w, i.e. $P_i(t_w) < P_n(t_w)$. The function max guarantees that it will not give a negative result. Therefore, the residue of τ_n at time $t_w + \epsilon$ is

$$r_n(t_w + \epsilon) = \max\left\{0, r_n(t_w) - \max\{0, \epsilon - \sum_{P_i(t_w) < P_n(t_w)} r_i(t_w)\}\right\}. \tag{18}$$

(3) For the dynamic response time $o_n(t)$, we know that $o_n(t)$ will continuously increase before τ_n finishes computation. Therefore, if τ_n has finished computation before t_w, i.e. $r_n(t_w) = 0$, we have that

$$o_n(t_w + \epsilon) = o_n(t_w) \tag{19}$$

On the other hand, if τ_n has NOT finished computation before t_w, we have that

$$o_n(t_w + \epsilon) = o_n(t_w) + \min\left\{r_n(t_w) + \sum_{P_i(t_w) < P_n(t_w)} r_i(t_w), \epsilon\right\} \tag{20}$$

where $r_n(t_w) + \sum_{P_i(t_w)<P_n(t_w)}$ denotes the extra delay before τ_n finishes computation. The function min guarantees that the increase of $o_n(t)$ will NOT exceed ϵ. Based on the above analysis, we can express $o_n(t)$ at time $t_w + \epsilon$ as

$$o_n(t_w + \epsilon) = o_n(t_w) + \mathsf{sgn}(r_n(t_w)) \min \left\{ r_n(t_w) + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), \epsilon \right\} \tag{21}$$

According to Equation (16), (18), and (20), the evolution of $[s_n(t), r_n(t), o_n(t)]$ from t_w to $t_w + \epsilon$ can be expressed as

$$\begin{aligned} s_n(t_w + \epsilon) &= s_n(t_w) - \epsilon \\ r_n(t_w + \epsilon) &= \max \left\{ 0, r_n(t_w) - \max\{0, \epsilon - \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w)\} \right\} \\ o_n(t_w + \epsilon) &= o_n(t_w) + \mathsf{sgn}(r_n(t_w)) \min \left\{ r_n(t_w) + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), \epsilon \right\}, \end{aligned} \tag{22}$$

In summary, Equation (15) and (22) constitute the dynamic timing model for the evolution of $Z(t)$ within a sub-interval $[t_w, t_{w+1}^-]$.

The dynamic timing model in Equation (15) and (22) can be implemented using Algorithm 1. Given the initial values of the state variables at t_w^-, i.e. $Z(t_w^-)$, and the task characteristics within the sub-interval, i.e., $\{C_n(t), T_n(t)\}_{n=1}^N$ for $t \in [t_w, t_{w+1}^-]$, we can use Algorithm 1 to obtain the evolution of the state variables from t_w^- to any time $t_w + \epsilon \in [t_w, t_{w+1}^-]$. The dynamic timing model for the evolution of $Z(t)$ within $[t_a, t_b]$ can be obtained by iteratively applying Algorithm 1 to all sub-intervals within $[t_a, t_b]$.

3.2.5 Scheduled Behaviors of Tasks

We demonstrate how to use the dynamic timing model to describe the scheduled behaviors of the real-time tasks. Consider $\Gamma = \{\tau_1, \cdots, \tau_N\}$, we first describe scheduled behavior of task τ_n from Γ. Within each sub-interval $[t_w, t_{w+1}^-]$, the scheduled behavior of task τ_n may go through three modes that will be indicated by a function $\Phi_n(t)$:

Algorithm 1: Model

/* when $\epsilon \in [0, L_w)$*/
Data: $t_w + \epsilon$, $Z(t_w^-)$, $\{C_n(t), T_n(t)\}_{n=1}^N$
Result: $Z(t_w + \epsilon)$

for *each task* $\tau_n \in \Gamma$ **do**
  /*the evolution of $Z(t)$ from t_w^- to t_w*/
  $s_n(t_w) \xleftarrow{\text{Equation (6)}} s_n(t_w^-)$;
  $r_n(t_w) \xleftarrow{\text{Equation (6)}} r_n(t_w^-)$;
  $o_n(t_w) \xleftarrow{\text{Equation (6)}} o_n(t_w^-)$;
  /*the evolution of $Z(t)$ from t_w to $t_w + \tau$*/
  $s_n(t_w + \epsilon) \xleftarrow{\text{Equation (7)}} s_n(t_w)$;
  $r_n(t_w + \epsilon) \xleftarrow{\text{Equation (7)}} r_n(t_w)$;
  $o_n(t_w + \epsilon) \xleftarrow{\text{Equation (7)}} o_n(t_w)$;
return $Z(t_w + \epsilon)$;

The preempted mode: the computation of the effective instance of τ_n is blocked by tasks with higher priorities at time t. This behavior is indicated by letting $\Phi_n(t) = 0.5$. The preempted mode starts from the beginning of the sub-interval t_w and finishes at $\min\{t_w + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), t_{w+1}^-\}$, where $t_w + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w)$ denotes the sum of the remaining execution time of all higher priority tasks;

The execution mode: the effective instance of τ_n is being computed by the CPU. The scheduled behavior is indicated by letting $\Phi_n(t) = 1$. The execution mode starts right after the preempted mode and finishes until the computation of the effect instance of τ_n completes, which equals $\min\{t_w + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), t_{w+1}^-\}$;

The free mode: the computation of the effective instance of τ_n has completed and new instance has not arrived. This behavior is indicated by letting $\Phi_n(t) = 0$. The free mode starts right after the execution mode and finishes till the end of the sub-interval.

In summary, the scheduled behavior of τ_n within each sub-interval $[t_w, t_{w+1}^-]$ can

be expressed as

$$\Phi_n(t) =$$

$$\begin{cases} 0.5, \\ t \in [\, t_w \,,\, \min\{t_w + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), t_{w+1}^-\}\,] \\ 1, \\ t \in (\, \min\{t_w + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), t_{w+1}^-\} \,,\, \min\{t_w + r_n(t_w) + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_w), t_{w+1}^-\}\,] \\ 0, \\ t \in (\, \min\{t_w + r_n(t_w) + \sum_{P_i(t_w)<P_n(t_w)} r_i(t_f), t_{w+1}^-\} \,,\, t_{w+1}^-\,]. \end{cases} \tag{23}$$

As it shows, the scheduled behavior of τ_n within each sub-interval $[t_w, t_{w+1}^-]$ can be described by the state vector $Z(t)$ within $[t_w, t_{w+1}^-]$. Applying the same methodology for all tasks in Γ, we can derive the scheduled behavior of the real-time system within $[t_w, t_{w+1}^-]$. As the sub-interval moves forward, the state vector $Z(t)$ will evolve according to the dynamic timing model in Algorithm 1. With $Z(t)$ evolving from t_a to t_b, we obtain the scheduled behavior of the real-time system over the time interval $[t_a, t_b]$.

3.3 Dynamic Schedulability Test

Real-time computing systems are becoming increasinly dynamic, and may operate in a fast changing environment, where complete specifications are not predictable at the design stage and/or operational requirements may adjust at system runtime. To guarantee normal operation of the real-time computing system in a dynamic environment, we introduce a new way to perform the schedulability analysis at runtime as follows:

Definition 3.3.1 *A* dynamic schedulability test *over a time interval* $[t_a, t_b]$ *checks if all task instances are able to meet their deadlines within* $[t_a, t_b]$.

As the starting time t_a increases, the time interval $[t_a, t_b]$ will slide forward. The length of the interval $(t_b - t_a)$ depends on how confident we are to predict the actual characteristics of tasks to perform the schedulability test.

Consider a set of tasks $\Gamma = \{\tau_1, \cdots, \tau_N\}$ running on the processor within $[t_a, t_b]$. Γ is schedulable within $[t_a, t_b]$ if and only if Γ is schedulable within each sub-interval $[t_w, t_{w+1}^-] \in [t_a, t_b]$, and Γ is schedulable within a sub-interval $[t_w, t_{w+1}^-]$ if and only if each individual task $\tau_n \in \Gamma$ is schedulable within $[t_w, t_{w+1}^-]$. The following theorem states the necessary and sufficient conditions for the schedulability of τ_n within a sub-interval $[t_w, t_{w+1}^-]$.

Theorem 3.3.2 *A task τ_n is schedulable within $[t_w, t_{w+1}^-]$ if and only if it satisfy ONE of the following conditions:*

1. $o_n(t_{w+1}^-) = T_n(t_{w+1}^-)$ *and* $r_n(t_{w+1}^-) = 0$;
2. $o_n(t_{w+1}^-) < T_n(t_{w+1}^-)$.

Proof 3.3.3 *If the dynamic response time of τ_n is equal to its relative deadline at time t_{w+1}^-, i.e. $o_n(t_{w+1}^-) = T_n(t_{w+1}^-)$, then the schedulability of τ_n within $[t_w, t_{w+1}^-]$ is satisfied if and only if the effective instance of τ_n has completed computation before time t_{w+1}^-, i.e. $r_n(t_{w+1}^-) = 0$.*

If the dynamic response time of τ_n is smaller than its relative deadline at time t_{w+1}^-, i.e. $o_n(t_{w+1}^-) < T_n(t_{w+1}^-)$, the schedulability of τ_n within $[t_w, t_{w+1}^-]$ is automatically guaranteed.

At any time t_a, given the task characteristics $\{C_n(t), T_n(t)\}_{n=1}^N$ for $t \in [t_a, t_b]$, we can use Algorithm 2 to perform the dynamic schedulability test over the time interval $[t_a, t_b]$. Algorithm 2 iteratively checks the schedulability of Γ within each sub-interval in the following ways: (1) first, at the beginning of any sub-interval, it calculates the end of the current sub-interval according to Equation (11), as shown in Lines 10 of

Algorithm 2; (2) then, it utilizes the dynamic timing model in Algorithm 1 to obtain the values of the state variables at the end of the current sub-interval, as indicated by Line 11; and (3) finally, it evaluates the schedulability of τ_n, where $n = 1, \cdots, N$, within $[t_w, t_{w+1}^-]$ according to Theorem 3.3.2, as shown in Lines $12 - 20$. To make the sub-interval propagates seamlessly within $[t_a, t_b]$, it assigns the starting time of the next sub-interval to be the ending time of the current sub-interval, as indicated by Line 20.

The variable $\mathrm{ds}_n[w]$ indicates the dynamic schedulability test result of τ_n within $[t_w, t_{w+1}^-]$: when τ_n is schedulable within $[t_w, t_{w+1}^-]$, $\mathrm{ds}_n[w] = 1$; otherwise, $\mathrm{ds}_n[w] = 0$. The set $\mathrm{DS}_n = \{\, ds_n[1],\, ds_n[2], \cdots\}$ contains the dynamic schedulability test results of τ_n within all sub-intervals that belong to $[t_a, t_b]$. The task τ_n is schedulable within $[t_a, t_b]$ if and only if $\min_w\{\mathrm{DS_n}\} = 1$. The task set Γ is schedulable within $[t_a, t_b]$ if and only if all individual tasks are dynamically schedulable within $[t_a, t_b]$, i.e. $\min_{1 \le n \le N}\{\min_w\{\mathrm{DS_n}\}\} = 1$.

3.4 Robustness of Real-time Computing Systems

In the operation of real-time computing systems, the actual task characteristics may often deviate from nominal task characteristics due to some unexpected online perturbations. This section presents a method to evaluate the robustness of the designed real-time computing systems to online perturbations.

3.4.1 Online Perturbation

We let $\{C_n^{\mathrm{nom}}(t), T_n^{\mathrm{nom}}(t)\}_{n=1}^N$ denote the nominal task characteristics known at the design phase, and $\{C_n(t), T_n(t)\}_{n=1}^N$ denote the actual task characteristics under online perturbations. We assume that the online perturbation is imposed on the computation time but NOT on the period, i.e.

$$T_n(t) = T_n^{\mathrm{nom}}(t), \;\; C_n(t) \neq C_n^{\mathrm{nom}}(t) \;\text{ for }\; n = 1, 2, ..., N. \tag{24}$$

This assumption is reasonable in control and robotics applications, where $T_n(t)$ represent sampling times that are often fixed. At time t, we define the (instantaneous) perturbations on computing times as follows:

Definition 3.4.1 *The* perturbations on computing times *are defined as a vector* $\mathcal{H}(t) = [\eta_1(t), ..., \eta_N(t)]$, *where* $\eta_n(t) = C_n(t) - C_n^{\text{nom}}(t)$ *for* $n = 1, 2, ..., N$.

The value of $\eta_n(t)$ can be either positive or negative. If $C_n(t) > C_n^{\text{nom}}(t)$, then $\eta_n(t)$ is positive.

The perturbation $\mathcal{H}(t)$ will accumulate over time and finally reflect on the state vectors. These effects will be captured by defining perturbations on the state variables. We let $\{s_n^{\text{nom}}(t), r_n^{\text{nom}}(t), o_n^{\text{nom}}(t)\}_{n=1}^N$ denote the state vector in the nominal case, and let $\{s_n(t), r_n(t), o_n(t)\}_{n=1}^N$ denote the state vector under accumulated perturbations. Since $T_n(t) = T_n^{\text{nom}}(t)$ for $n = 1, 2, ..., N$, the release time of each task instance in the nominal case is the same as these in the actual case. Thus, according to Definition 3.2.1, we know that the dynamic inter-release time of each task instance in the nominal case is the same as that in the actual case, i.e.

$$s_n(t) = s_n^{\text{nom}}(t) \tag{25}$$

which, together with Equation (10), implies that

$$t_w = t_w^{\text{nom}} \quad L_w = L_w^{\text{nom}} \quad t_{w+1} = t_{w+1}^{\text{nom}}. \tag{26}$$

On the other hand, since $C_n(t) \neq C_n^{\text{nom}}(t)$, we know that it may take less or more time for the instance of τ_n finishes computation. Hence, the dynamic response time of each task instance in the nominal case is different from these in the actual case, i.e.

$$o_n(t) \neq o_n^{\text{nom}}(t). \tag{27}$$

We define the perturbations on the dynamic response time as follows:

Definition 3.4.2 *The* perturbations on the dynamic response time *is defined as a vector* $\mathcal{E}(t) = [e_1(t), ..., e_N(t)]$*, where* $e_n(t)$ *denotes the strength of the perturbation on* $o_n(t)$*, i.e.*

$$e_n(t) = o_n(t) - o_n^{\text{nom}}(t) \tag{28}$$

3.4.2 Maximum Tolerable Perturbation

According to the above analysis, we know that at any time t, the accumulated perturbation of $\mathcal{H}(t)$ imposed on the real-time system is reflected $\mathcal{E}(t) = \{e_1(t), \cdots, e_N(t)\}$. In particular, the total perturbations imposed on one task τ_n at time t can be expressed as $e_n(t)$. We are interested in finding the maximum total perturbations $e_n(t)$ that can be tolerated by a single task τ_n without sacrificing the schedulability of τ_n. We have the following claims.

Claim 3.4.3 τ_n *is schedulable within* $[t_w, t_{w+1}^-]$ *under perturbations* $e_n(t)$ *if and only if the following condition is satisfied:*

$$e_n(t_{w+1}^-) \leq T_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) - o_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) \tag{29}$$

Proof 3.4.4 *According to the definition of perturbation in Equation (28), we know that the perturbation* $e_n(t)$ *can be expressed as follows*

$$\begin{aligned} e_n(t_{w+1}^-) &= o_n(t_{w+1}^-) - o_n^{\text{nom}}(t_{w+1}^-) \\ &= o_n(t_{w+1}^-) - T_n^{\text{nom}}(t_{w+1}^-) - o_n^{\text{nom}}(t_{w+1}^-) + T_n^{\text{nom}}(t_{w+1}^-) \end{aligned} \tag{30}$$

Moreover, since no perturbation is imposed on the period $T_n(t)$*, we have that*

$$\begin{aligned} T_n^{\text{nom}}(t_{w+1}^-) &= T_n(t_{w+1}^-) \\ o_n^{\text{nom}}(t_{w+1}^-) &= o_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) \\ T_n^{\text{nom}}(t_{w+1}^-) &= T_n(\{t_{w+1}^{\text{nom}}\}^-) \end{aligned} \tag{31}$$

where Equation (24) and (26) is applied. Using Equation (31) in Equation (32), we have that

$$e_n(t_{w+1}^-) = o_n(t_{w+1}^-) - T_n(t_{w+1}^-) - o_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) + T_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) \tag{32}$$

According to Theorem 3.3.2, we know that when the real-time system is schedulable under the perturbation, the following condition must be satisfied

$$o_n(t_{w+1}^-) - T_n(t_{w+1}^-) \leq 0 \tag{33}$$

which implies that

$$e_n(t_{w+1}^-) \leq T_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) - o_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) \tag{34}$$

Hence, the claim is proved.

3.4.3 Robustness Measure

We introduce a measure of robustness B_R that quantifies the tolerance of a real-time scheduling algorithm to uncertain perturbations to the computing times of tasks within $[t_a, t_b]$. A real-time scheduling algorithm with a larger value for B_R is more robust than a real-time scheduling algorithm with smaller values for B_R.

Definition 3.4.5 *We define a* measure of robustness $B_R(w)$ *over the sub-interval window* $[t_w, t_{w+1}^-]$ *where* $w = 1, 2, ...$ *as the least upper bound on the tolerable perturbations for all task instances expiring at* $t_w + L_w$*, i.e.*

$$B_R(w) = \min_{1 \leq n \leq N}(T_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) - o_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-)) \tag{35}$$

We define the measure of robustness B_R *over time interval* $[t_a, t_b]$ *as the minimum value of* $B_R(w)$ *i.e.*

$$B_R = \min_w B_R(w). \tag{36}$$

Claim 3.4.6 *Within $[t_a, t_b]$, the nominal design of an acyclic task set under a real-time scheduling algorithm is schedulable under any perturbation of a strength less than B_R.*

Proof 3.4.7 *Suppose an arbitrary task τ_n suffers the perturbation $e_n(t_{w+1}^-)$ at the end of a sub-interval window $[t_w, t_{w+1}^-]$. Since we have that $e_n(t_{w+1}^-) \leq B_R \leq B_R(w) \leq T_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-) - o_n^{\text{nom}}(\{t_{w+1}^{\text{nom}}\}^-)$, Claim 3.4.3 is satisfied and τ_n is schedulable to the perturbation. The above proof holds for any task within any fixed priority window that belongs to $[t_a, t_b]$. Hence, the nominal design is schedulable under any perturbation of a strength less than B_R.*

At any time t_a, if we input the nominal task characteristics $\{T_n^{\text{nom}}(t)\}_{n=1}^N$ and $\{C_n^{\text{nom}}(t)\}_{n=1}^N$ to Algorithm 1, we can obtain the evolution of the nominal state variables $\{s_n^{\text{nom}}(t), r_n^{\text{nom}}(t), o_n^{\text{nom}}(t)\}_{n=1}^N$ from t_a to t_b by iteratively applying the dynamic timing model in Algorithm 1. Moreover, the right hand side of Equation (35) is computed at t_a by using the nominal state vector.

3.5 Numeric Simulation

In this section, we use numeric simulations to show that the dynamic timing model can accurate represent the scheduled behavior of tasks on the processor and the dynamic schedulability test is a necessary and sufficient test.

3.5.1 Simulation Setup

We consider a set of tasks $\Gamma = \{\tau_1, \tau_2, \tau_3\}$ running simultaneously on the processor. At the design phase, we assume $\{\tau_1, \tau_2, \tau_3\}$ have the following characteristics

$$\begin{aligned} &[\ T_1^{\text{nom}}(t),\ T_2^{\text{nom}}(t),\ T_3^{\text{nom}}(t)\ ] = [\ 15.4,\ 20.8,\ 30.3\ ]\text{ms}, \\ &[\ C_1^{\text{nom}}(t),\ C_2^{\text{nom}}(t),\ C_3^{\text{nom}}(t)\ ] = [\ \ 4\ \ ,\ \ 4\ \ ,\ \ 4\ \ ]\text{ms} \end{aligned} \tag{37}$$

Tasks are scheduled under the RMS algorithm. According to Section 3.2.2 and the nominal task characteristics in Equation (37), we have the task priorities as

$$P_1(t) = 1 \quad P_2(t) = 2 \quad P_3(t) = 3 \tag{38}$$

Moreover, we assume that online perturbations $\mathcal{H}(t) = [\eta_1(t), \eta_2(t), \eta_3(t)]$ will happen on the computing time $\{C_n^{\text{nom}}(t)\}_{n=1}^3$ within time interval $[10, 13]$s. $\{\eta_n(t)\}_{n=1}^3$ is a stochastic processes with their value at each point in time as a random variable uniformly distributed within $[-1.5, 4]$ms, $[-1, 4]$ms and $[-1, 2]$ms. Given the perturbation $\mathcal{H}(t)$, we have the actual task characteristics within $[10, 13]$s as

$$\begin{aligned} [\, T_1(t), T_2(t), T_3(t) \,] &= [\quad T_1^{\text{nom}}(t) \quad , \quad T_2^{\text{nom}}(t) \quad , \quad T_3^{\text{nom}}(t) \quad] \\ [\, C_1(t), C_2(t), C_3(t) \,] &= [\, C_1^{\text{nom}}(t) + \eta_1(t),\ C_2^{\text{nom}}(t) + \eta_2(t),\ C_3^{\text{nom}}(t) + \eta_3(t) \,]. \end{aligned} \tag{39}$$

3.5.2 Verification of Dynamic Timing Model

To verify the dynamic timing model, we compare the scheduled behavior of Γ derived from the dynamic timing model in Equation (23) with that simulated using TrueTime [20]. TrueTime is one of the most commonly used software tools for research on real-time control.

We run the simulation from 0 to 10s using the nominal task characteristics in Equation (37), through both TrueTime 1.5 implemented in MATLAB and the dynamic timing model. Figure 5 shows the results of the two different methods within $[9.29, 9.63]$s. By comparison, we see that the scheduled behaviors generated by TrueTime 1.5 and the dynamic timing model are identical for most of the time. The identical part indicates that the dynamic timing model can be used to describe the scheduled behavior of tasks as precisely as TrueTime. However, the scheduled behaviors generated by TrueTime 1.5 and the dynamic timing model are not identical for $\Phi_2(t)$ when $t \in [9.3016, 9.3056]$s and for $\Phi_3(t)$ when $t \in [9.5788, 9.5828]$s. Further exploration shows that the differences are due to jitters caused by the numerical inaccuracy in TrueTime 1.5 implemented in MATLAB, as illustrated in the upper half

of Figure 5. As a simulation tool, TrueTime 1.5 inevitably has truncation errors that accumulate with numerical integration. Since the dynamic timing model presented in this paper is based on mathematical equations, the system behavior at time t can be determined by evaluating functions without using numerical integration. Hence no jitters are observed from the lower half of Figure 5.

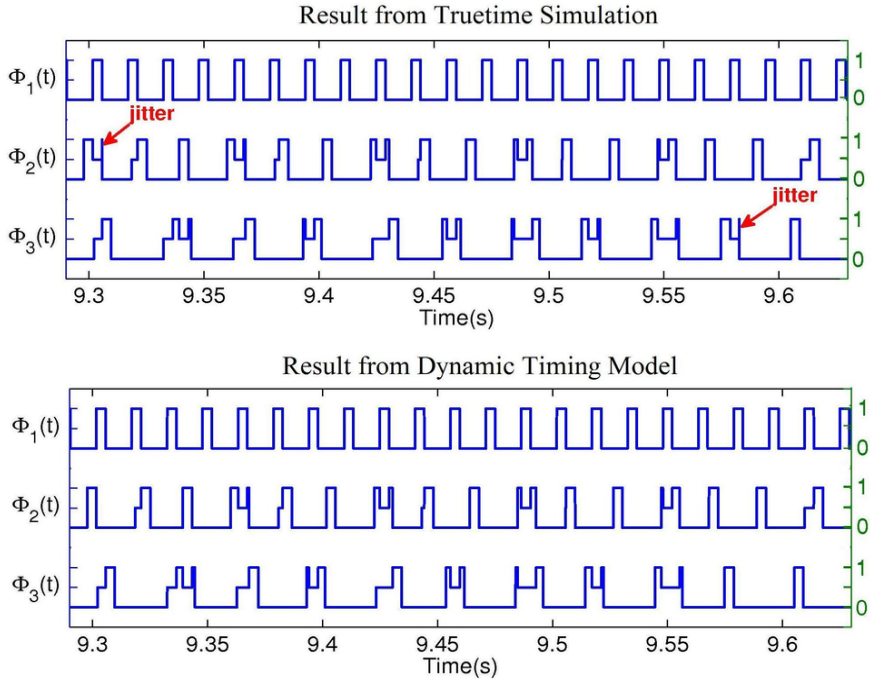


Figure 5: Scheduled behaviors of Γ within $[9.29, 9.63]$ seconds. Upper figure produced by TrueTime. Lower figure produced by the dynamic timing model. Jitters marked by arrows.

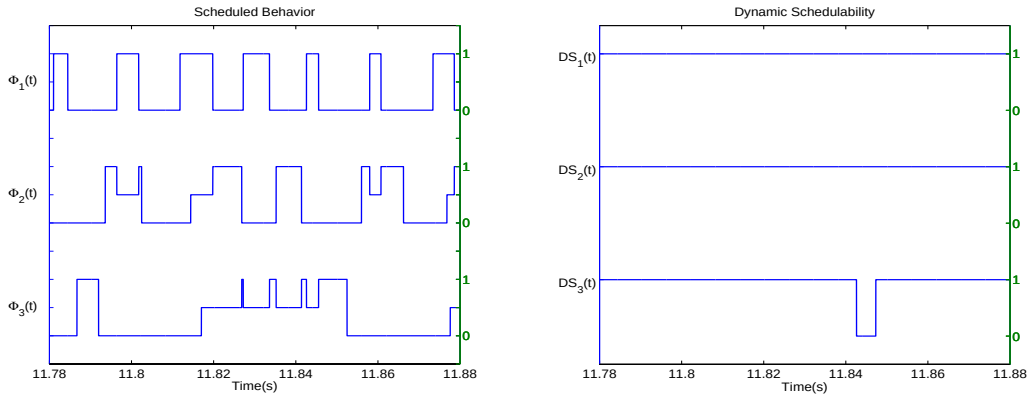
3.5.3 Computational Cost Analysis

Our dynamic timing model also computes faster than Truetime Simulation. To verify this computational advantage, we first generate the scheduled behavior of a given task set through both two methods as dynamic timing model and Truetime simulation, and then evaluate the run-time of each method. We consider three different tasks with $[T_1, T_2, T_3] = [6, 5, 4]$ms and $[C_1, C_2, C_3] = [2, 2, 2]$ms, and they are scheduled under the RMS scheduling algorithms. We generate the scheduled behavior of three tasks within $[0, 20]$s through both two methods. The experiment is performed on a

MacBook computer with Processor 2.26 GHz Intel Core 2 Duo, and Memory 4GB 1067MHz DDR3. Since Truetime is written in C++ Mex [20], we also write the dynamic timing model in the same way as Truetime. We run each method 1000 times and then calculate the average run-time of each method. The experiments show that the average run-time of generating scheduled behavior through Truetime is 1.2067s, and that through our dynamic timing model is 2.2473×10^{-4}s. As it shows, our dynamic timing model is more than 5000 times faster than Truetime simulation in this case.

3.5.4 Dynamic Schedulability Test

The dynamic schedulability test can be used online to re-check the schedulability of the real-time computing system whenever task characteristics change. In this section, we will perform the dynamic schedulability test within the time interval $[10, 13]$ when the online perturbation happens.



(a) Scheduled Behavior of Γ (b) Dynamic Schedulability Test of Γ

Figure 6: Three Tasks under RMS scheduling

To check the schedulability, we use the simulation tool TrueTime plot the scheduled behavior of tasks in Figure 6.(a), and the result of dynamic schedulability tests is shown in Figure 6.(b). By closely observing that the value of $\Phi_3(t)$, we can see that the value of $\Phi_3(t)$ does not fall back to zero before its deadline at $t = 11.8473$s. This

observation implies that τ_3 fails to finish computation by its deadline. In Figure 6.(b), $\mathrm{DS}_n(t)$ denotes the dynamic schedulability test result of τ_n at time t. $\mathrm{DS}_n(t) = 1$ denotes that the effective instance of τ_n at time t has missed its deadline, and $\mathrm{DS}_n(t) = 0$ denotes that the effective instance of τ_n at time t has NOT missed its deadline. As we can see, $\mathrm{DS}_3(t) = 0$ when $t \in [11.817, 11.8473$s. This observation indicates that τ_3 has missed its deadline within $[11.817, 11.8475]$s. Therefore, the observation in Figure 6.(b) exactly match that in Figure 6.(a). This match implies that our dynamic schedulability analysis can accurately identify unschedulable task instances.

3.5.5 Robustness of real-time scheduling

We demonstrate that the scheduling algorithm with a higher B_R is more robust to the perturbations. Consider two different scheduling algorithms as the RMS algorithm and the EDF algorithm. When the tasks are scheduled under the RMS algorithm, we calculate the value of B_R within $[10, 13]$s to be 8.8 according to Definition 3.4.5. When the tasks are scheduled under the EDF algorithm, we calculate the value of B_R within $[10, 13]$s to be 11.4. Since the system using the EDF algorithm has a higher measure of robustness as compared with the system using the RMS algorithm, we conclude that the former is more robust to the perturbations considered. Indeed, under the same perturbation $\mathcal{H}(t)$, our dynamic schedulability test has confirmed that the real-time task set under the EDF algorithm is still schedulable, but is not schedulable under the RMS algorithm.

Algorithm 2: Dynamic Schedulability Test

/*Schedulability of Γ within $[t_a, t_b]$ */
Data: t_a, t_b, $Z(t_a^-)$, $\{C_n(t), T_n(t)\}_{n=1}^{N}$
Result: $\{\mathrm{DS_n}\}_{\mathrm{n=1}}^{\mathrm{N}}$

$t_w = t_a$;
for *each* $\tau_n \in \Gamma$ **do**
    $\mathrm{DS_n} = [\,]$;
/*check each fixed priority window*/
while $t_w < t_b$ **do**
    for *each* $\tau_n \in \Gamma$ **do**
        if $s_n(t_w^-) == 0$ **then**
            $s_n(t_w) = T_n(t_w)$;
        else
            $s_n(t_w) = s_n(t_w^-)$;
    /* The length of the current fixed priority window L_w */
    $t_{w+1} = t_w + \min\{s_1(t_w), ..., s_N(t_w), t_b - t_w\}$;
    /* State Vector at the end of the current sub-interval */
    $Z(t_{w+1}^-) = \mathrm{Model}(t_w, t_{w+1}^-, Z(t_w^-), \{C_n(t), T_n(t)\}_{n=1}^{N})$;
    /* Schedulability within the current sub-interval */
    for *each* $\tau_n \in \Gamma$ **do**
        if $s_n(t_{w+1}^-) == 0$ **then**
            if $o_n(t_{w+1}^-) < T_n(t_{w+1}^-)$ **then**
                $\mathrm{ds_n} = 1$;
            else
                $\mathrm{ds_n} = 0$;
        else
            $\mathrm{ds_n} = 1$;
        $\mathrm{DS_n} = \{\mathrm{DS_n}, \mathrm{ds_n}\}$;
    $w = w + 1$;
return $\{\mathrm{DS_n}\}_{\mathrm{n=1}}^{\mathrm{N}}$;

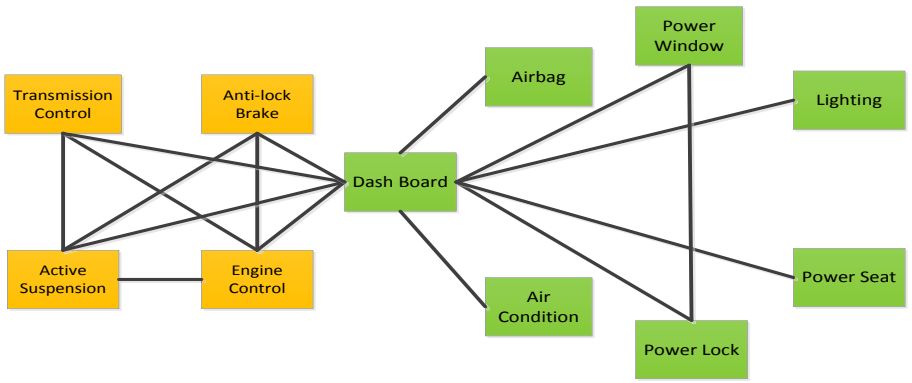
CHAPTER IV

CAN BUS BASED MPC DESIGN

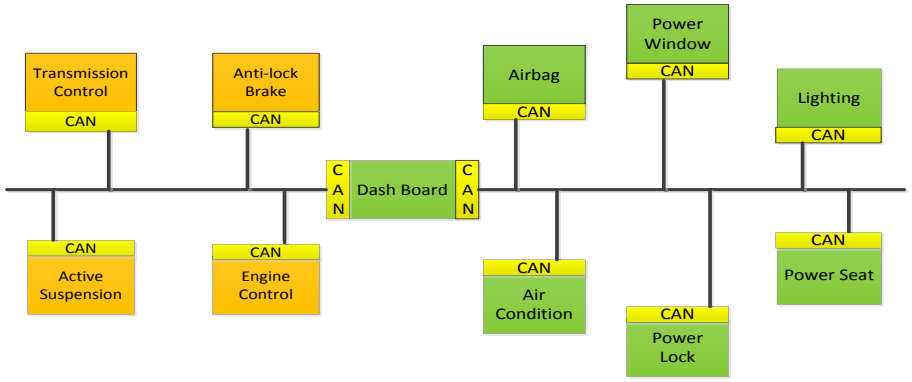
The controller area network (CAN bus) is a communication network developed by automobile industries for reducing wiring complexity among electronic control units in vehicles. Figure 7 shows an example of connections among electronic control units in vehicles. As we can see in Figure 7.(a), before the CAN bus was invented, every pair of electronic control units has to connect with each other through point-to-point wiring. Therefore, the complexity of wiring grows exponential as the number of electronic control unit increases and the whole system soon becomes too complicated to maintain. On the other hand, as we can see in Figure 7.(b), the CAN bus provides an inexpensive, durable network that helps multiple devices communicate with one another. An advantage to this is that electronic control units can have a single CAN interface rather than connections to every device in the system. This decreases overall cost and weight in automobiles.

The CAN bus has an distinguished feature of supporting real-time communication. Each message on the CAN bus is assigned a priority and the transmission of messages is scheduled according to their priorities. This feature makes the CAN bus particularly suitable for safety-critical systems with stringent time constraints, such as robotics, aircraft, medical equipment, and industrial automation. The CAN-based control system is a system in which feedback control loops are closed via the CAN bus. The integration of the CAN bus and feedback control loops can bring many benefits, such as high system flexibility and low maintenance cost. Reference [6] proposed the design of a self-triggered controller on the CAN bus and showed that this design

methodology reduces the bandwidth usage in the CAN-based control system. Reference [55] studied how to improve the aggregated control performance by effectively allocating bandwidth of the CAN bus among multiple control loops. Reference [38] discussed the design of the CAN bus-based manipulator arm. These works assume that the delay between sampling and actuation are either constant or known.



(a) Point to Point Connection



(b) Connection Based on the CAN bus

Figure 7: Electronic Control Units inside Automobiles

4.1 CAN-based Control System

We study a CAN-based control system, as shown in Figure 8. Each feedback control loop utilizes the CAN bus to pass sampled data from sensors to embedded controllers, and to pass control commands from embedded controllers to actuators. The sensors, embedded controllers, and actuators are connected to the CAN bus through the CAN chips. Hence we can call them sensor nodes, embedded controller nodes, and actuator

nodes on the CAN bus. We simplify the design so that each feedback control loop has one sensor node, one embedded controller node, and one actuator node. This is not to be considered as only allowing single-input single-output systems because multiple sensors can be integrated into a sensor node, and multiple actuators can be integrated into an actuator node.

We configure and implement the CANed system in Figure 8 according to the following rules:

- At the sensor node, a user specified function samples the state of the plants, and generate a sensor message that contains sampled data;
- At the embedded controller node, upon reception of a sensor message, a user specified function extracts sampled data, computes the control law, and generate a control message that contains control signals;
- At the actuator node, upon reception of a control message, a user specified function extracts control signals from the control message and issue the control on the plant .

The above configuration implies a **causality constraint** between sensor and control messages as follows: in each feedback control loop, a sensor message must be transmitted BEFORE the embedded controller starts computing the control law. A control message can only be transmitted AFTER the control law is computed.

Real-time scheduling of messages on the CAN introduces temporal non-determinism in feedback control loops. Such temporal non-determinism is often reflected by time-varying response time between the moment when the sensors take measurements, and the moment when the actuators take actions. Such delay may severely degrade the control performance if not well accounted in the design. Our goal is to derive an analytical model that is able to accurately predict response-time for CANed systems.

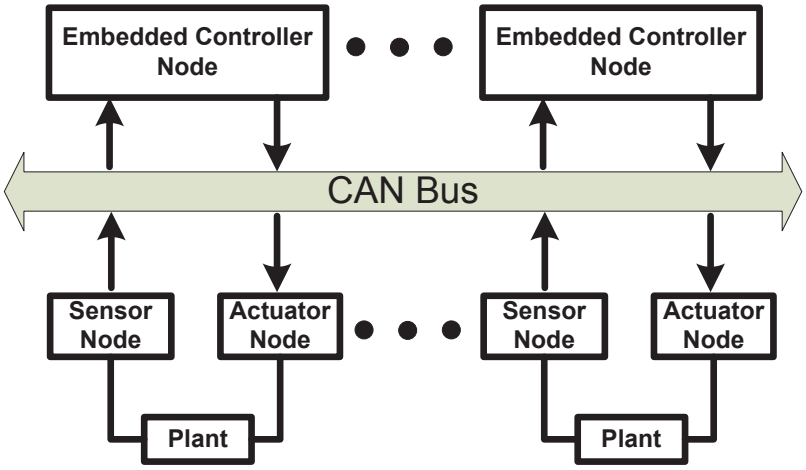


Figure 8: Multiple Feedback Loops Sharing CAN

4.2 Message Chains

The causality constraint requires that the transmission of a sensor message is followed by the computation of the embedded controller, which is followed by the transmission of a control message. This process iteratively repeats in the procedure of controlling the plant. We call each iteration of this process, beginning from the sampling of sensor and ending at the actuation, an *instance*, and then define the above process for any n-th feedback control loop as a *message chain* τ_n. Thus, each message chain τ_n is composed of recurring instances. We use the index $k = 1, 2, ...$ to indicate each of the recurring instances in τ_n for the n-th loop. Let the k-th instance of τ_n be denoted by $\tau_n[k]$. According to the above description, we know that $\tau_n[k]$ starts at the k-th sampling instant $\alpha_n[k]$.

$\tau_n[k]$ contains two sub-messages as $\tau_n^1[k]$ and $\tau_n^2[k]$. $\tau_n^1[k]$ represents the sensor message, and $\tau_n^2[k]$ represents the control message. Figure 9 shows an example of a message chain τ_n. The horizontal line represents the progression of time, $\alpha_n[k]$ represents the time instant when the sensors take measurement, $I_n^1[k]$ represents the amount of time for the sensor node to sample plants and prepare $\tau_n^1[k]$, $C_n^1[k]$ represents the transmission duration of $\tau_n^1[k]$, $\beta_n[k]$ represents the time instant when the

transmission of $\tau_n^1[k]$ is completed, $I_n^2[k]$ represents the amount of time for the controller node to compute MPC and prepare $\tau_n^2[k]$, $C_n^2[k]$ represents the transmission duration of $\tau_n^2[k]$, $\gamma_n[k]$ represents the time instant when the transmission of $\tau_n^2[k]$ is completed, and $T_n[k]$ represents the sampling interval between $\alpha_n[k]$ and $\alpha_n[k+1]$. Note there may exist some general-purpose messages that are not related to control, but share the CAN bus with the feedback control loops. These general-purpose messages can also be represented by message chains. For example, we can let a message chain τ_j to represent a general purpose message by choosing $I_j^2[k] = 0$ and $C_j^2[k] = 0$.

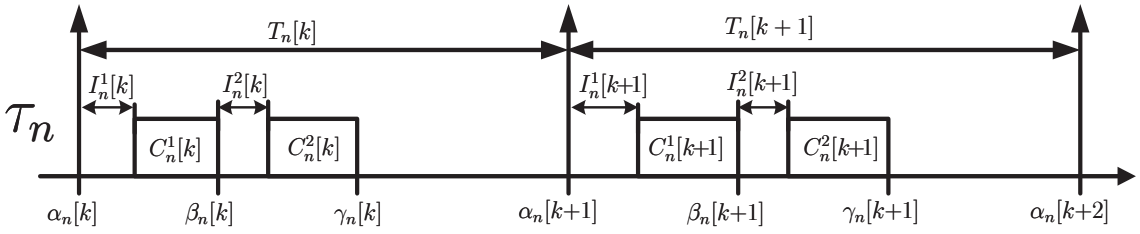


Figure 9: An example message chain τ_n if no other messages are sharing CAN

Since only one message can be transmitted on the CAN bus at a time, $\tau_n^1[k]$ and $\tau_n^2[k]$ in $\tau_n[k]$ may not be transmitted immediately after they are generated. Instead, they have to compete with other messages for access to the CAN bus, under the CSMA/BA arbitration scheme. The priority of $\tau_n[k]$ can be represented by $P_n[k]$. Since each sub-message $\tau_n^1[k]$ and $\tau_n^2[k]$ in $\tau_n[k]$ may have its own priority, we have

$$P_n[k] = \begin{cases} P_n^1[k] & \text{when } \tau_n^1[k] \text{ is transmitted} \\ P_n^2[k] & \text{when } \tau_n^2[k] \text{ is transmitted} \end{cases} \tag{40}$$

where $P_n^1[k]$ and $P_n^2[k]$ represent the priorities (identifier fields) of $\tau_n^1[k]$ and $\tau_n^2[k]$, respectively.

Since we want to model the scheduled behaviors of message chains at any time t, we define the message characteristics in continuous time domain as follows

Definition 4.2.1 *For any message chain τ_n, an instance $\tau_n[k]$ is active at time t if and only if it starts before t and its next instance starts after t, i.e. $\alpha_n[k] < t <$*

$\alpha_n[k+1]$. *At any time t,* τ_n *has only one active instance denoted as* $\tau_n(t)$*, i.e.*

$$\text{if } \alpha_n[k] \leq t < \alpha_n[k+1] \quad \text{then } \tau_n(t) = \tau_n[k] \tag{41}$$

Definition 4.2.2 *At any time t, we define* $\tau_n^1(t)$ *and* $\tau_n^2(t)$ *as the first and second sub-messages in* $\tau_n(t)$*, i.e.*

$$\text{if } \alpha_n[k] \leq t < \alpha_n[k+1], \tau_n^1(t) = \tau_n^1[k] \text{ then } \tau_n^2(t) = \tau_n^2[k] \tag{42}$$

Based on the above definition, we can represent the message characteristics in Figure 9 as follows: $I_n^1(t)$ and $I_n^2(t)$ as the time spent for preparing $\tau_n^1(t)$ and $\tau_n^2(t)$, $C_n^1(t)$ and $C_n^2(t)$ as the transmission duration of $\tau_n^1(t)$ and $\tau_n^2(t)$, $T_n(t)$ as the sampling interval of $\tau_n(t)$, and $P_n(t)$ as the priority of $\tau_n(t)$. Therefore, each message chain τ_n can be characterized by the tuple $\{T_n(t), I_n^1(t)+C_n^1(t)+I_n^2(t)+C_n^2(t), P_n(t)\}$. These notations are summarized in Table 1.

Table 1: Characteristics of a message chain τ_n

$\tau_n(t)$	Active instance of the message chain τ_n at time t
$\tau_n^1(t)$ and $\tau_n^2(t)$	First and second sub-messages in $\tau_n(t)$
$T_n(t)$	Sampling interval of $\tau_n(t)$
$P_n(t)$	Priority of $\tau_n(t)$
$C_n^1(t)$ and $C_n^2(t)$	Transmission duration of $\tau_n^1(t)$ and $\tau_n^2(t)$
$I_n^1(t)$ and $I_n^2(t)$	Time spent for preparing $\tau_n^1(t)$ and $\tau_n^2(t)$

4.3 Hybrid Timing Model

The problem of scheduling message chains on a CAN bus shares some similarity with the problem of task scheduling on a processor. However, scheduling message chains on a CAN bus is a more complex problem. First, messages on the CAN bus are un-preemptive while tasks in real-time computing are preemptive. Moreover, messages on the CAN bus are subject to causality constraints while tasks in real-time computing are independent. Such increased complexity requires significant extensions and we will show that a hybrid timing model is a convenient way to describe the timing of events on the CAN bus.

4.3.1 State Vector

To model the dynamics of scheduling a set of message chains $\{\tau_1, \cdots, \tau_N\}$ on the CAN bus, we introduce a state vector $Z(t) = [S(t), R(t), O(t)]$. Each component in the state vector $Z(t)$ has the same meaning as that introduced in Definition 3.2.1 $\sim$ Definition 3.2.3. For the purpose of readability, we re-write the definitions in the following part.

Definition 4.3.1 *The dynamic inter-release time is defined as* $S(t) = [s_1(t), \cdots, s_N(t)]$, *where* $s_n(t)$, *for* $n = 1, 2, \cdots, N$, *denotes how long after* t *the next instance of* τ_n *is released.*

Definition 4.3.2 *The residue is defined as* $R(t) = [r_1(t), ..., r_N(t)]$, *where* $r_n(t)$, *for* $n = 1, 2, ..., N$, *denotes the remaining execution time of the active instance of* τ_n after *time* t.

Definition 4.3.3 *The dynamic response time is defined as* $O(t) = [o_1(t), ..., o_N(t)]$, *where* $o_n(t)$, *for* $n = 1, 2, ..., N$, *denotes how much time has elapsed before the active instance of* τ_n *finishes execution.*

The state vector $Z(t)$ evolves continuously most of the time, except when two "special" events happen. One special event is that a message chain τ_n releases a new instance. When this event happens, $[s_n(t), r_n(t), o_n(t)] \in Z(t)$ is reset to the characteristics of the new instance of τ_n. The other special event is that a different message chain starts to transmit its messages on the CAN bus. When this event happens, the evolution dynamics of $Z(t)$ will switch discontinuously.

4.3.2 Evolution of State Vector

Since $Z(t)$ exhibits both continuous and discrete dynamic behaviors, the natural choice for describing the evolution of $Z(t)$ is to use a hybrid automaton defined as follows [78, 84].

Definition 4.3.4 *A hybrid automaton describes the dynamics of scheduling* $\{\tau_1, \cdots, \tau_N\}$. *It is a collection* $H = \{Q, Z, F, \text{Dom}, \text{Edge}, \text{Guard}, \text{Reset}\}$ *where*

- $Q = \{q_0, \cdots, q_N\}$ *is a set of modes, where the mode* q_0 *indicates that no message is being transmitted over the CAN bus and the mode* q_i $(1 \leq i \leq N)$ *indicates that* τ_i *is transmitting its messages over the CAN bus;*

- $Z(t) = [S(t), R(t), O(t)] \in \mathbb{R}^{3N}$ *is a continuous state vector as defined above;*

- $F : Q \times Z \to \mathbb{R}^{3N}$ *is the flow map. For each* $q_i \in Q$, $F(q_i, Z)$ *describes the continuous evolution of* Z *in the mode* q_i*;*

- $\text{Dom} : Q \to 2^Z$ *is the domain of modes. For each* $q_i \in Q$, $\text{Dom}(q_i)$ *identifies a set of* Z *that evolves continuously in the mode* q_i*;*

- $\text{Edge} : \text{Edge} \subseteq Q \times Q$ *is a set of edges. Each* $(q_i, q_j) \in \text{Edge}$ *indicates that a discrete transition from the mode* q_i *to the mode* q_j *is possible;*

- $\text{Guard} : \text{Edge} \to 2^Z$ *is the jump condition. For each* $(q_i, q_j) \in \text{Edge}$, $\text{Guard}(q_i, q_j)$ *identifies a set of* Z *that can trigger a discrete transition from the mode* q_i *to the mode* q_j*;*

- $\text{Reset} : \text{Edge} \times Z \to 2^Z$ *is the reset map. For each* $(q_i, q_j) \in \text{Edge}$, $\text{Reset}(q_i, q_j, Z)$ *describes the value to which* Z *is reset during a discrete transition from the mode* q_i *to the mode* q_j*;*

Figure 10 demonstrates a directed graph representation of the hybrid automaton H when $N = 2$. The graphical representation of H with other values of N can be easily constructed using the same methodology. As shown in Figure 10, the vertices represent modes and the arrows represent edges. Within each vertex the flow map and the domain set are indicated. Along each edge the jump condition and the reset map are shown.

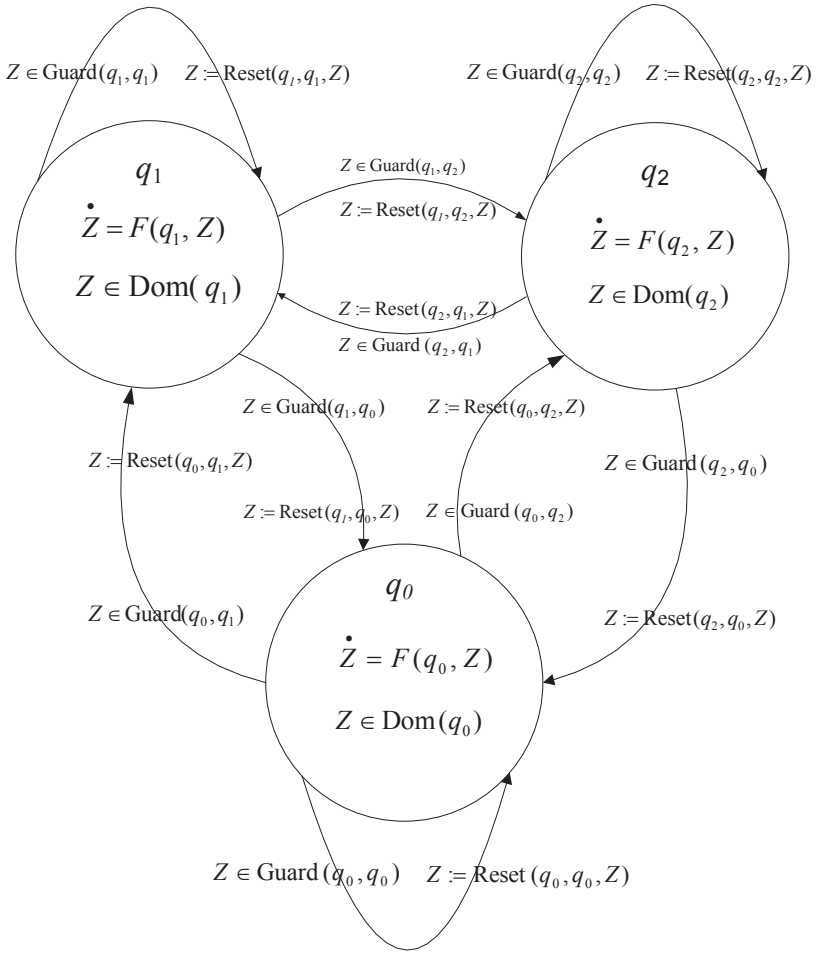


Figure 10: Graphical representation of the hybrid automaton H when $N = 2$

4.3.3 The Hybrid Automata

In this section, we derive the expressions of F, Dom, Edge, Guard and Reset for the hybrid automaton H, respectively. For the ease of derivation, we classify message chains at any time t into two sets $G_{\text{rdy}}(t)$ and $G_{\text{prep}}(t)$ according to their current state $Z(t)$. Note that $G_{\text{rdy}}(t) \bigcup G_{\text{prep}}(t) = \{1, \cdots, N\}$

Definition 4.3.5 *$G_{\text{rdy}}(t)$ is defined as a set of message chains, which have messages ready for transmission at time t. $G_{\text{rdy}}(t)$ can be expressed as*

$$G_{\text{rdy}}(t) = \left\{ n \;\middle|\; \begin{array}{c} I_n^2(t) + C_n^2(t) < r_n(t) \leq C_n^1(t) + I_n^2(t) + C_n^2(t) \\ \text{or} \quad 0 < r_n(t) \leq C_n^2(t) \end{array} \right\} \tag{43}$$

where the first condition specifies a message chain τ_n with $\tau_n^1(t)$ ready for transmission, and the second condition specifies a message chain τ_n with $\tau_n^2(t)$ ready for transmission.

Definition 4.3.6 *$G_{\text{prep}}(t)$ is defined as a set of message chains, which are preparing sub-messages at time t. $G_{\text{prep}}(t)$ can be expressed as*

$$G_{\text{prep}}(t) =$$
$$\left\{ n \;\middle|\; \begin{array}{c} C_n^1(t) + I_n^2(t) + C_n^2(t) < r_n(t) \leq I_n^1(t) + C_n^1(t) + I_n^2(t) + C_n^2(t) \\ \text{or} \;\; C_n^2(t) < r_n(t) \leq I_n^2(t) + C_n^2(t) \end{array} \right\} \tag{44}$$

where the first condition specifies a message chain τ_n with $\tau_n^1(t)$ being prepared, and the second condition specifies a message chain τ_n with $\tau_n^2(t)$ being prepared.

Flow Map F. We discuss the continuous evolution of $Z(t)$ in any mode $q_i \in Q$. Since $Z(t)$ consists of three components as $Z(t) = [S(t), R(t), O(t)]$, we will discuss the continuous evolution of each component respectively. We use $\Delta t > 0$ to denote an arbitrarily small change in time. First, we study the continuous evolution of $S(t)$ in the mode q_i. Consider an element $s_n(t) \in S(t)$, we have

$$\dot{s}_n(t) = -1 \tag{45}$$

Applying equation (45) for $n = 1, \cdots, N$, we will have the continuous evolution of $S(t)$ in the mode q_i. Next, we study the continuous evolution of $R(t)$ in the mode q_i. Consider an element $r_n(t) \in R(t)$.

$$\dot{r}_n(t) = \begin{cases} -1 & n = i \text{ or } n \in G_{\text{prep}}(t) \\ 0 & \text{otherwise} \end{cases} \tag{46}$$

Applying equation (46) for $n = 1, \cdots, N$, we will have the continuous evolution of $R(t)$ in the mode q_i. Finally, we study the continuous evolution of $O(t)$ in the mode q_i. Consider an element $o_n(t) \in O(t)$. According to Definition 4.3.3, we know that $o_n(t)$ will continue increasing until $\tau_n(t)$ finishes. Therefore, we have $o_n(t + \Delta t) = o_n(t) + \text{sgn}(r_n(t))\,\Delta t$, where sgn is a funciton such that $\text{sgn}(x) = 1$ if $r_n(t) > 0$, $\text{sgn}(x) = 0$ if $x = 0$, and $\text{sgn}(x) = 1$ if $x < 0$. Hence,

$$\dot{o}_n(t) = \lim_{\Delta t \to 0} \frac{o_n(t + \Delta t) - o_n(t)}{\Delta t} = \text{sgn}(r_n(t)) \tag{47}$$

Applying equation (47) for $n = 1, \cdots, N$, we will have the continuous evolution of $O(t)$ in the mode q_i. In summary, for each $q_i \in Q$, $F(q_i, Z)$ describes the continuous evolution of Z in the mode q_i,

$$F(q_i, Z) = \begin{bmatrix} \dot{S}(t) \\ \dot{R}(t) \\ \dot{O}(t) \end{bmatrix} = \begin{bmatrix} \dot{s}_1(t) & \cdots & \dot{s}_N(t) \\ \dot{r}_1(t) & \cdots & \dot{r}_N(t) \\ \dot{o}_1(t) & \cdots & \dot{o}_N(t) \end{bmatrix} = \begin{bmatrix} -1 & \cdots & -1 \\ -\mathcal{X}_A(1) & \cdots & -\mathcal{X}_A(N) \\ \text{sgn}(r_1(t)) & \cdots & \text{sgn}(r_N(t)) \end{bmatrix}. \tag{48}$$

$A = \{ i \} \cup G_{\text{prep}}(t)$ is a set of indices of message chains. $\mathcal{X}_A(n)$ is an indicator function such that $\mathcal{X}_A(n) = 1$ if $n \in A$ and $\mathcal{X}_A(n) = 0$ if $n \notin A$.

Domain of Modes. Our discussion on the domain of modes in Q can be classified into two cases depending on different types of modes as follows.

Case 1: in the mode q_0, the state vector Z will continuously evolve as long as the following two conditions are both satisfied: no new instance of $\{\tau_1, \cdots, \tau_N\}$ starts

and no message is being transmitted. The first condition is satisfied if we have that

$$\min_{1\leq n\leq N}\{S(t)\} > 0 \tag{49}$$

where Definition 4.3.1 is applied. The second condition is satisfied if $G_{\text{rdy}}(t)$ is an empty set, i.e.

$$\text{Card}\left(G_{\text{rdy}}(t)\right) = 0 \tag{50}$$

where Card$(\cdot)$ is a cardinality function that measures the number of elements in a set. Therefore, we have the domain of the mode q_0 as

$$\text{Dom}(q_0) = \left\{ Z(t) \,|\, \min_{1\leq n\leq N}\{S(t)\} > 0 \,\text{and}\, \text{Card}\left(G_{\text{rdy}}(t)\right) = 0 \right\} \tag{51}$$

Case 2: in the mode q_i where $1 \leq i \leq N$, the state vector Z will continuously evolve as long as the following two conditions are both satisfied: no new instance of $\{\tau_1, \cdots, \tau_N\}$ starts and τ_i is transmitting its messages over the CAN bus at time t. The first condition is satisfied if we have that

$$\min_{1\leq n\leq N}\{S(t)\} > 0 \tag{52}$$

where Definition 4.3.1 is applied. The second condition is satisfied if we have that

$$i \in G_{\text{rdy}}(t) \tag{53}$$

where Definition 4.3.5 is applied. Therefore, we have the domain of the mode q_i where $1 \leq i \leq N$ as

$$\text{Dom}(q_i) = \left\{ Z(t) \,|\, \min_{1\leq n\leq N}\{S(t)\} > 0 \text{ and } i \in G_{\text{rdy}}(t) \right\} \tag{54}$$

In summary, for each $q_i \in Q$, Dom(q_i) identifies a set of Z that evolves continuously in the mode q_i, i.e.

$$\text{Dom}(q_i) = \begin{cases} \{\ Z(t) \,|\, \min_{1\leq n\leq N}\{S(t)\} > 0 \,\text{and}\, \text{Card}\left(G_{\text{rdy}}(t)\right) = 0\ \} & \text{when } i = 0 \\ \{\ Z(t) \,|\, \min_{1\leq n\leq N}\{S(t)\} > 0 \text{ and } i \in G_{\text{rdy}}(t)\ \} & \text{when } 1 \leq i \leq N \end{cases} \tag{55}$$

Edge. According to the definition of Q, we know that the transition between any modes is possible. Therefore, we have that

$$\text{Edge} = Q \times Q \tag{56}$$

Jump Conditions. For each edge $(q_i, q_j) \in \text{Edge}$, we discuss the jump condition $\text{Guard}(q_i, q_j)$. Our discussion can be classified into four cases as follows.

Case 1: an edge (q_i, q_i). This transition is triggered when a new instance of τ_i starts, i.e.

$$\text{Guard}(q_i, q_i) = \{\, Z(t) \,|\, s_i(t) = 0\} \tag{57}$$

Case 2: an edge (q_i, q_j) where $i \neq j$ and $1 \leq i, j \leq N$. This transition is triggered when a message of τ_i finishes transmission and a message of τ_j takes over the CAN bus for transmission. Thus, we have

$$\text{Guard}(q_i, q_j) = \{\, Z(t) \,|\, i \notin G_{\text{rdy}}(t) \text{ and } j = \underset{n \in G_{\text{rdy}}(t)}{\text{argmin}}\, P_n(t)\} \tag{58}$$

where the first condition $i \notin G_{\text{rdy}}(t)$ indicates that τ_i has no message ready for transmission and the second condition $j = \text{argmin}_{n \in G_{\text{rdy}}(t)} P_n(t)$ indicates that τ_j has the highest priority among all messages ready for transmission.

Case 3: an edge (q_i, q_0) where $1 \leq i \leq N$. This transition is triggered when a message of τ_i finishes transmission and no other message is ready for transmission at time t. Thus, we have

$$\text{Guard}(q_i, q_0) = \{\, Z(t) \,|\, i \notin G_{\text{rdy}}(t) \text{ and } \text{Card}\,(G_{\text{rdy}}(t)) = 0\} \tag{59}$$

Case 4: an edge (q_0, q_j) where $1 \leq j \leq N$. This transition is triggered when a message of τ_i is transmitted after the CAN bus has been idle for a while.

$$\text{Guard}(q_0, q_j) = \{Z(t) \,|\, \text{Card}\,(G_{\text{rdy}}(t)) > 0 \text{ and } j = \underset{n \in G_{\text{rdy}}(t)}{\text{argmin}}\, P_n(t)\} \tag{60}$$

In summary, for each $(q_i, q_j) \in$ Edge, Guard(q_i, q_j) identifies a set of Z that triggers a discrete transition from the mode q_i to the mode q_j, i.e.

$$\text{Guard}(q_i, q_j) =$$

$$\begin{cases} \{\, Z(t) \,|\, s_i(t) = 0\} & \text{when } i = j \\ \{\, Z(t) \,|\, i \notin G_{\text{rdy}}(t) \text{ and } j = \underset{n \in G_{\text{rdy}}(t)}{\text{argmin}}\ P_n(t)\} & \text{when } 0 < i \neq j \leq N \\ \{\, Z(t) \,|\, i \notin G_{\text{rdy}}(t) \text{ and Card}\,(G_{\text{rdy}}(t)) = 0\} & \text{when } 0 < i \leq N, j = 0 \\ \{Z(t) \,|\, \text{Card}\,(G_{\text{rdy}}(t)) > 0 \text{ and } j = \underset{n \in G_{\text{rdy}}(t)}{\text{argmin}}\, P_n(t)\} & \text{when } i = 0, 0 < j \leq N \end{cases} \tag{61}$$

Reset Map. We use t and t^+ to denote the time right before the reset and the time right after the reset. First, we consider Reset(q_i, q_i, Z) for an edge (q_i, q_i). As discussed in equation (60), this transition is triggered when a new instance of τ_i starts at time t. Consider the reset of $\{s_n(t), r_n(t), o_n(t)\}_{n=1}^N$ in Reset(q_i, q_i, Z).

Case 1: $n \neq i$. In this case, τ_n does not have new instance arrives at time t. Hence, the state variables of τ_n hold their values during the transition, i.e.

$$\text{if } \ s_n(t) > 0 : s_n(t^+) = s_n(t),\ r_n(t^+) = r_n(t),\ o_n(t^+) = o_n(t) \tag{62}$$

Case 2: $n = i$ and the old instance of τ_i has finished its execution before time t, i.e. $d_i(t) = 0$ and $r_i(t) = 0$. In this case, the state variables of τ_i is reset to the characteristics of the new instance, i.e.

$$\text{if } s_i(t) = r_i(t) = 0 \text{: } s_i(t^+) = T_i(t),\ r_i(t^+) = I_i^1(t) + C_i^1(t) + I_i^2(t) + C_i^2(t),\ o_i(t^+) = 0. \tag{63}$$

Case 3: $n = i$ but the old instance of τ_i has not finished its execution before time t, i.e. $s_i(t) = 0$ and $r_i(t) > 0$. In this case, we dismiss the new instance of τ_i. Therefore, $r_i(t)$ and $o_i(t)$ hold their values from t to t^+, $s_i(t^+)$ is reset to $T_i(t)$ of the dismissed instance, i.e.

$$\text{if } \ s_i(t) = 0, r_i(t) > 0 : s_i(t^+) = T_i(t),\ r_i(t^+) = r_i(t),\ o_i(t^+) = o_i(t). \tag{64}$$

Equation (62), (63) and (64) constitute the reset map $\text{Reset}(q_i, q_i, Z)$ for an edge (q_i, q_i). Next, we discuss $\text{Reset}(q_i, q_j, Z)$ for an edge (q_i, q_j), where $i \neq j$. During this transition, the flow map will change but the state vector $Z(t)$ remains constant. Thus, the reset map $\text{Reset}(q_i, q_j, Z)$ is an identity map such that

$$\text{Reset}(q_i, q_j, Z) = Z(t) \tag{65}$$

In summary, for each $(q_i, q_j) \in \text{Edge}$, $\text{Reset}(q_i, q_j, Z)$ describes the value to which Z is reset during a discrete transition from the mode q_i to the mode q_j. Thus, we have that

$$\text{Reset}(q_i, q_j, Z) = \begin{bmatrix} s_1(t) - (\text{sgn}(s_1(t)) - 1)\, T_1(t) & \cdots & s_N(t) - (\text{sgn}(s_N(t)) - 1)\, T_N(t) \\ r_1(t) - (\text{sgn}(s_1(t) + r_1(t)) - 1)\, C_1(t) & \cdots & r_N(t) - (\text{sgn}(s_N(t) + r_N(t)) - 1)\, C_N(t) \\ o_1(t)\, \text{sgn}(s_1(t) + r_1(t)) & \cdots & o_N(t)\, \text{sgn}(s_N(t) + r_N(t)) \end{bmatrix}. \tag{66}$$

From the above derivation of $\{F, \text{Dom}, \text{Edge}, \text{Guard}, \text{Reset}\}$, we can have the following claim

Claim 4.3.7 *For the hybrid automaton H, at any time point t, given initial state $[Q(t), Z(t)]$ and message chains $\{T_n(t+\epsilon), I_n^1(t+\epsilon) + C_n^1(t+\epsilon) + I_n^2(t+\epsilon) + C_n^2(t+\epsilon), P_n(t+\epsilon)\}_{n=1}^N$, there exists a unique trajectory of $[Q(t+\epsilon), Z(t+\epsilon)]$ for $\epsilon > 0$.*

Proof 4.3.8 *The claims will hold automatically once we prove that the hybrid automaton H is non-blocking and deterministic,*

First, we prove that the hybrid automaton H is non-blocking. According to the automaton theory, a hybrid automaton is called non-blocking if for all reachable states at which the continuous evolution is impossible a discrete transition is possible. Consider any mode q_i. In the mode q_i, the continuous evolution is impossible if the state vector $Z(t) \notin Dom(q_i)$. According to Equation (55), we know that all reachable

states at which the continuous evolution is impossible constitute a set that equals to the complement of $Dom(q_i)$, i.e.

$$\mathrm{Dom}^c(q_i) = \begin{cases} \{\ Z(t) \mid \min\limits_{1\le n\le N}\{S(t)\} = 0\} \cup \{\ Z(t) \mid \mathrm{Card}\,(G_{\mathrm{rdy}}(t)) > 0\} & \text{when } i = 0 \\ \{\ Z(t) \mid \min\limits_{1\le n\le N}\{S(t)\} = 0\ \} \cup \{\ Z(t) \mid i \notin G_{\mathrm{rdy}}(t)\ \} & \text{when } 1 \le i \le N \end{cases} \tag{67}$$

Moreover, in the mode q_i, all reachable states at which the discrete transition is possible constitutes a set $\bigcup_{j=0}^{N} \mathrm{Guard}(q_i, q_j)$. According to Equation (61), we have that

$$\mathrm{Dom}^c(q_i) = \bigcup_{j=0}^{N} \mathrm{Guard}(q_i, q_j) \tag{68}$$

which implies that for every $Z(t) \in \mathrm{Dom}^c(q_i)$, the discrete transition is possible.

Next, we show the hybrid automaton H is deterministic. According to the automaton theory, a hybrid automaton is called deterministic if and only if (1) each discrete transition has a unique destination, and (2) whenever a discrete transition is possible continuous evolution is impossible. The first condition can be checked through Figure 10. The second condition is satisfied as $\mathrm{Dom}^c(q_i) = \bigcup_{j=0}^{N} \mathrm{Guard}(q_i, q_j)$.

Based on the above claim, we can represent the hybrid timing model of the CAN-based control system as follows: for any $\epsilon > 0$

$$[Q(t+\epsilon\,|\,t), Z(t+\epsilon\,|\,t)] = H\left([Q(t), Z(t)], \{T_n(t+\epsilon), I_n^1(t+\epsilon) + C_n^1(t+\epsilon) + I_n^2(t+\epsilon) + C_n^2(t+\epsilon), P_n(t+\epsilon)\}_{n=1}^{N}\right) \tag{69}$$

where $[Q(t+\epsilon|), Z(t+\epsilon|t)]$ for any $\epsilon > 0$ represents the state vector predicted from $[Q(t), Z(t)]$ at time t.

4.4 State Observer Design

At each embedded controller node, we want to use the hybrid timing model to predict delays and timing constraints for MPC. The prediction requires the knowledge of the

state vector $[Q(t), Z(t)]$. But the values of the state vector may not be known. In this section, we will discuss how to estimate the state vector based on events that can be observed on the CAN bus.

4.4.1 Estimation of $Z(t)$

As the first step in estimating $Z(t)$, we need to figure out what kind of information each embedded controller node can directly observe from the CAN bus. As discussed in [23], CAN chips can generate an interrupt to the host processor whenever an message is received from the CAN bus. Therefore, we can easily design an interrupt handler on the host processor to observe the receiving time of $\tau_n^1[k]$ and $\tau_n^2[k]$, which corresponds to $\beta_n[k]$ and $\gamma_n[k]$ as shown in Figure 9. But there is no direct way to measure $\alpha_n[k]$. Note that the CAN bus utilizes a broadcast scheme for message transmission. The embedded controller node in each feedback loop can not only receive messages within its own control loop, but also messages from other feedback control loops. Therefore, each embedded controller node has complete information of $\{\beta_n[k], \gamma_n[k]\}_{n=1}^N$ for all message chains $\{\tau_1, \cdots, \tau_N\}$ on the CAN bus. Based on the above observations, we propose an algorithm of estimating the value of $\alpha_n[k]$ as follows

$$\hat{\alpha}_n[k] = \min\{\hat{\alpha}_n[k-1]+T_n[k-1],\ \beta_n[k]-C_n^1[k]-I_n^1[k]\} \tag{70}$$

where $\hat{\alpha}_n[k-1]$ is an estimation from the previous observation.

At any time t, given $\{\hat{\alpha}_n[k], \beta_n[k], \gamma_n[k]\}_{n=1}^N$, each embedded controller node can estimate the state vector $\hat{Z}(t) = \{\hat{s}_n(t), \hat{o}_n(t), \hat{r}_n(t)\}_{n=1}^N$ as follows:

$$\hat{s}_n(t) = \hat{\alpha}_n[k] + T_n[k] - t, \tag{71}$$

in which $\hat{\alpha}_n[k] + T_n[k]$ denotes the time instant when $\tau_n[k+1]$ starts;

$$\hat{o}_n(t) = \begin{cases} t - \hat{\alpha}_n[k] & \tau_n^2[k] \text{ NOT received} \\ \gamma_n[k] - \hat{\alpha}_n[k] & \tau_n^2[k] \text{ received} \end{cases}, \tag{72}$$

which implies that the delay will not increase if $\tau_n^2[k]$ has finished transmission before t; and

$$\hat{r}_n(t) =$$

$$\begin{cases} I_n^1[k]+C_n^1[k]+I_n^2[k]+C_n^2[k]-\min\{\, t-\hat{\alpha}_n[k], I_n^1[k]\,\} & \text{if } \tau_n^1[k] \text{ and } \tau_n^2[k] \text{ NOT received} \\ I_n^2[k] + C_n^2[k] - \min\{t - \beta_n[k], I_n^2[k]\} & \text{if } \tau_n^1[k] \text{ received, but } \tau_n^2[k] \text{ NOT} \\ 0 & \text{if } \tau_n^1[k] \text{ and } \tau_n^2[k] \text{ received} \end{cases} \tag{73}$$

In summary, at any time t, each embedded controller node can estimate the state vector $\hat{Z}(t)$ using Equations (70) $\sim$ (73).

4.4.2 Estimation of $Q(t)$

According to the introduction of Q in Definition 4.3.4, we know that

$$\hat{Q}(t)=$$

$$\begin{cases} q_n & \text{if } \hat{r}_n(t) \in (I_n^2[k] + C_n^2[k], C_n^1[k] + I_n^2[k] + C_n^2[k]) \bigcup (0, C_n^2[k]) \\ q_0 & \text{otherwise} \end{cases} \tag{74}$$

where the first condition specifies a message chain τ_n that is being transmitted on the CAN bus at time t. Since the CAN bus can only transmit one message at a time, there exists only one message chain τ_n that can meet the first condition.

From Equation (74), we know that $\hat{Q}(t)$ can be derived from $\hat{R}(t) \in \hat{Z}(t)$. Therefore, the estimation of $\hat{Q}(t)$ solely depends on $\hat{Z}(t)$.

4.4.3 Convergence of Estimation

In this section, we show that the estimation $[\hat{Q}(t), \hat{Z}(t)]$ will gradually converge to the actual values $[Q(t), Z(t)]$ as time t propagates.

As we have discussed before, $[\hat{Q}(t), \hat{Z}(t)]$ depends on $\{\hat{\alpha}_n[k], \beta_n[k], \gamma_n[k]\}_{n=1}^N$. Since $\{\beta_n[k], \gamma_n[k]\}_{n=1}^N$ can be directly observed from the CAN bus, the accuracy of estimating $[\hat{Q}(t), \hat{Z}(t)]$ is actually determined by the accuracy of estimating $\hat{\alpha}_n[k]$.

Therefore, $[\hat{Q}(t), \hat{Z}(t)]$ converges to $[Q(t), Z(t)]$ as t propagates, if and only if $\hat{\alpha}_n[k]$ converges to $\alpha_n[k]$ as k increases. We denote the estimation error between $\hat{\alpha}_n[k]$ and $\alpha_n[k]$ as

$$\epsilon_n[k] = \hat{\alpha}_n[k] - \alpha_n[k] \text{ for any } k \geq 0 \tag{75}$$

To prove $\hat{\alpha}_n[k]$ converging to $\alpha_n[k]$ as k increases, we need to show that

Claim 4.4.1 *The estimation error $\epsilon_n[k]$ is non-negative and monotonically decreasing as k increases, i.e.*

$$\epsilon_n[0] \geq \epsilon_n[1] \geq \cdots \geq \epsilon_n[k] \geq \epsilon_n[k+1] \geq \cdots \geq 0. \tag{76}$$

Proof 4.4.2 *First, we prove that the estimation error is non-negative, i.e. $\epsilon_n[k] \geq 0$ for any $k \geq 0$. Each message may not be transmitted immediately after it is ready. Thus, we have that*

$$\alpha_n[k] + I_n^1[k] \leq \beta_n[k] - C_n^1[k] \text{ for any } k \geq 0 \tag{77}$$

where the left hand side represents the time when a message $\tau_n^1[k]$ is ready for transmission and the right hand side represents the time when $\tau_n^1[k]$ actually starts to transmit on the CAN bus. According to Equation (70) and (77), we know that

$$\hat{\alpha}_n[0] = \beta_n[0] - C_n^1[0] - I_n^1[0] \geq \alpha_n[0] \tag{78}$$

which implies $\epsilon_n[0] \geq 0$. Moreover, we have that

$$\hat{\alpha}_n[0] + T_n[0] \geq \alpha_n[0] + T_n[0] = \alpha_n[1]. \tag{79}$$

Moreover, according to Equation (77), we have that

$$\beta_n[1] - C_n^1[1] - I_n^1[1] \geq \alpha_n[1] \tag{80}$$

Therefore, based on Equation (70), (79), and (80), we have that

$$\hat{\alpha}_n[1] = \min\{\hat{\alpha}_n[0] + T_n[0],\ \beta_n[1] - C_n^1[1] - I_n^1[1]\} \geq \alpha_n[1] \tag{81}$$

which implies that $\epsilon_n[1] \geq 0$. *By proof induction, we can easily show that* $\epsilon_n[k] \geq 0$ *for any* $k \geq 0$.

Next, we prove that the estimation error $\epsilon_n[k]$ *is monotonically decreasing as* k *increases, i.e.* $\epsilon_n[k] \geq \epsilon[k+1]$. *According to Equation (70), we have that*

$$\hat{\alpha}_n[k+1] \leq \hat{\alpha}_n[k] + T_n[k] \tag{82}$$

which implies that

$$\hat{\alpha}_n[k+1] - \hat{\alpha}_n[k] \leq T_n[k] = \alpha_n[k+1] - \alpha_n[k] \tag{83}$$

Hence, we have that

$$\hat{\alpha}_n[k] - \alpha_n[k] \geq \hat{\alpha}_n[k+1] - \alpha_n[k+1] \tag{84}$$

Therefore, $\epsilon[k] \geq \epsilon[k+1]$ *for any* $k \geq 0$ *is proved.*

4.5 MPC Design for CAN-based Control System

Model predictive control (MPC) is an control algorithm that has been widely used in the processing industry such as oil refinery and chemical plants. The basic idea of MPC algorithm is to iteratively use a process model of the physical system to predict and optimize future system behaviors. At each control instant, an MPC algorithm will compute an trajectory of future control signal that will optimize predicted plant output within a finite time horizon. Only the first part of the future control signal will be applied to the plant and the entire calculation is repeated at subsequent control instant.

4.5.1 Basic Principles

For any n-th feedback control loop in Figure 8, we assume the plant is an independent, multiple input multiple output, and linear time-invariant system

$$\begin{aligned} \dot{x}_n(t) &= Ax_n(t) + Bu_n(t) \\ y_n(t) &= Cx_n(t), \end{aligned} \tag{85}$$

where $u_n(t)$ is the control command, $y_n(t)$ is the plant output, $x_n(t)$ is the plant state, and A, B and C are state matrix of proper dimension. The system setup in Section 4.1 implies that: (1) Each MPC controller only generates one control command within each sampling interval. The resulting control command is applied on the plant and remains constant until the next sampling interval; and (2) time delay exists between the moment when the sensor takes measurements, and the moment when the actuator implements the control command. Based on the above implications, we know that the control command $u_n(t)$ in Equation (85) must be a piece-wise constant function

$$u_n(t) = \mu_n[k],\ t \in [\alpha_n[k] + \delta_n[k], \alpha_n[k+1] + \delta_n[k+1]) \tag{86}$$

where $\alpha_n[k]$ is the k-th sampling instant of the sensor, $\mu_n[k]$ is a control command generated by MPC controller within the sampling interval $[\alpha_n[k], \alpha_n[k+1]\,)$, and $\delta_n[k]$ is the time delay between sampling instant $\alpha_n[k]$ and corresponding actuation instant. Equation (85) and (86) constitutes the process model for each MPC controller node in Figure 8.

At any time t_0, we assume that an estimate of the current state $\hat{x}_n(t_0)$ is known from a filtering algorithm (for example, Kalman filter) based on the past sensor messages. Given the $\hat{x}_n(t_0)$, the goal of MPC design is to find an optimal future trajectory of control command $u_n(t_0+\tau)$ that brings the future plant output $y_n(t_0+\tau)$ as close as possible to a reference trajectory $\gamma_n(t_0+\tau)$, within a prediction horizon $\tau \in [0, T_p]$, where T_p is the length of prediction. This problem can be mathematically formulated as

$$\text{Given } \hat{x}_n(t_0),\ \min_{u_n(t_0+\tau)} J(u_n(t_0+\tau)) \tag{87}$$

where the cost function $J(u_n(t_0+\tau))$ is typically given by

$$J(u_n(t_0+\tau)) =$$
$$\int_0^{T_p} (\gamma_n(t_0+\tau) - y_n(t_0+\tau))^T V (\gamma_n(t_0+\tau) - y_n(t_0+\tau)) + \dot{u}_n(t_0+\tau)^T W \dot{u}_n(t_0+\tau)\, d\tau \tag{88}$$

V and W are diagonal weight matrices. The first term on the right hand side represents the difference between the future plant output and the reference trajectory within the prediction horizon, and the second term on the right hand side represents the control penalty. Note that in Equation (88), $y_n(t_0 + \tau)$ for $\tau \in [0, T_p]$ must be predicted as a function of $\hat{x}_n(t_0)$ and $u_n(t_0 + \tau)$ for $\tau \in [0, T_p]$ through the process model in Equation (85) and (86).

4.5.2 Design Challenges

Accurate prediction of delays $\delta_n[k]$ is very important to MPC design. From the perspective of real-time scheduling, $\delta_n[k]$ is the response time between the moment when a task instance arrives and the moment when a task instance finishes execution. Analyzing the accurate value of $\delta_n[k]$ is very challenging because it is varying with respect to task instances under the real-time scheduling. Most of existing work in real-time scheduling focuses on analyzing the worst-case response time of each task, i.e. $\max_{k>0}\delta_n[k]$. However, using the worst-case response time will result in inaccurate process model, which will severely degrade system performance. Figure 11 shows an example of MPC performance under inaccurate predictions of $\delta_n[k]$. The inaccurate prediction of $\delta_n[k]$ is chosen as worst-case response time. The solid line represents the plant output $y_n(t)$ and the dashed line represents the reference trajectory $\gamma_n(t)$. As we can see, using an inaccurate $\delta_n[k]$ will lead to an unreliable process model, which severely degrades the performance of MPC.

4.5.3 MPC Design

In this section, we will solve the MPC design problem for the CAN-based control system. For simplicity, we only consider MPC design for the n-th feedback control loop in Figure 8. MPC design for other feedback loops can be obtained through the same procedures.

As discussed in Section 4.5.2, $\delta_n[k]$ is needed for MPC design. Based the hybrid

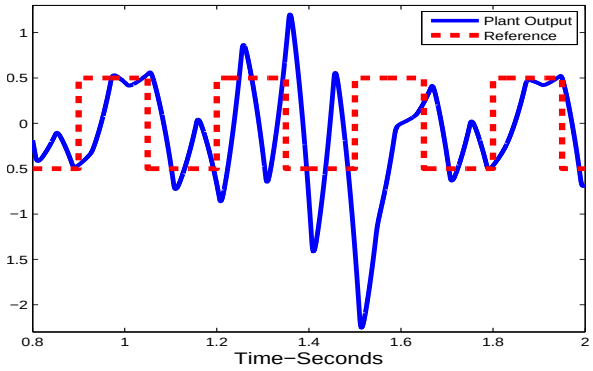


Figure 11: MPC design based on the worst-case response time

timing model H, we can perform online prediction of $\delta_n[k]$ as follows. First, at any time t_0, we estimate the state vector $[\hat{Q}(t_0), \hat{Z}(t_0)]$ by observing events on the CAN bus, as discussed in Section 4.4. Then, we predict the future trajectory of $[\hat{Q}(t_0 + \tau \,|\, t_0), \hat{Z}(t_0 + \tau \,|\, t_0)]$ within the prediction horizon $\tau \in [0, T_p]$ as

$$\begin{gathered}[\hat{Q}(t_0 + \tau|t_0), \hat{Z}(t_0 + \tau|t_0)] = \\ H([\hat{Q}(t_0), \hat{Z}(t_0)], \{T_n, I_n^1 + C_n^1 + I_n^2 + C_n^2, P_n\}_{n=1}^N (t_0 + \tau))\end{gathered} \tag{89}$$

where H represents the hybrid timing model of the CAN bus, and $\{T_n, I_n^1 + C_n^1 + I_n^2 + C_n^2, P_n\}_{n=1}^N (t_0 + \tau)$ are characteristics of message chains within the prediction horizon. Finally, according to Definition 4.3.3, we know that the delay between sampling and actuation in the n-th feedback control loop can be expressed as

$$\delta_n[k_0 + k] = \hat{o}_n(\hat{\alpha}_n[k_0] + \sum_{j=k}^{k_0+k-1} T_n[j] \,|\, t) \tag{90}$$

where $\hat{o}_n(t_0 + \tau|t_0)$ is an element that belongs to $\hat{Z}(t_0 + \tau|t_0)$, and $\delta_n[k_0 + k]$ represents delays within the prediction horizon. Note that Equation (89) and (90) are predictions performed online. Therefore, the predicted delays can continuously update according to changes in messages.

At every sampling instant $t_0 = \alpha_n[k_0]$, given the estimated state $\hat{x}_n(t_0)$ of the plant and the estimated state $[\hat{Q}(t_0), \hat{Z}(t_0)]$ of the CAN bus, we can reformulate the

MPC design as the following optimization problem

$$\text{Given } \hat{x}_n(t_0) \text{ and } [\hat{Q}(t_0), \hat{Z}(t_0)], \quad \min_{u_n(t_0+\tau)} J(u_n(t_0+\tau)) \tag{91}$$

s.t.

(1)

$$u_n(t_0+\tau) \in \mathcal{U}, \;\; x_n(t_0+\tau) \in \mathcal{X}, \tag{91.a}$$

(2)

$$\begin{aligned} \dot{x}_n(t) &= Ax_n(t) + Bu_n(t) \\ y_n(t) &= Cx_n(t) \end{aligned} \tag{91.b}$$

(3)

$$u_n(t) = \mu_n[k],\ t \in [\alpha_n[k]+\delta_n[k], \alpha_n[k+1]+\delta_n[k+1]) \tag{91.c}$$

(4)

$$[\ \hat{Q}(t_0+\tau \,|\, t_0),\ \hat{Z}(t_0+\tau \,|\, t)\ ] = H([\hat{Q}(t), \hat{Z}(t)], \{T_n, I_n^1+C_n^1+I_n^2+C_n^2, P_n\}_{n=1}^N(t_0+\tau)) \tag{91.d}$$

(5)

$$\delta_n[k_0+k] = \hat{o}_n(\alpha_n[k_0] + \sum_{j=k_0}^{k_0+k-1} T_n[j] \,|\, t) \tag{91.e}$$

where Equation (91.a) represents the constraints on the control command and the plant states, Equation (91.b) and (91.c) represents the physical plant in the process model, and Equation (91.d) and (91.e) represents timing constraints induced by the CAN bus. Note that the physical plant and the CAN bus timing model are coupled through the delay $\delta_n[k]$ in Equation (91.c) and (91.e).

4.6 Solving MPC Design

In this section, we solve the MPC design in Equation (87). Since the time delay $\delta_n[k]$ is variable for different k, the standard MPC methods from linear time-invariant discrete-time system cannot be applied. Therefore, we apply a two step approach to

perform the MPC design in the presence of the variable computational time delay. In the first step, we assume that $u(t)$ is continuously differentiable. Based on this assumption, we solve a continuous MPC design by utilizing Equation (91.a) and (91.b), to find out an optimal continuous control signal. In the second step, the resulting optimal continuous control signal is discretized according to the timing constraints in Equation (91.c), (91.d), and (91.e), to obtain optimal discrete control signals.

4.6.1 Continuous MPC Design

In the first step, we assume that the control command $u_n(t)$ is a continuously differentiable function. According to Equation (91.c) and $t_0 = \alpha_n[k_0]$, we know that

$$u_n(t_0 + \tau) = u_n(\alpha_n[k_0] + \tau) = \mu_n[k_0 - 1] \text{ for } \tau \in [0, \delta_n[k_0]) \tag{92}$$

where $\mu_n[k_0 - 1]$ is the control command that has been computed and fixed before t_0. Therefore, we are only concerned with the control signal within $[t_0 + \delta_n[k_0], t_0 + T_p]$, i.e.

$$u_n(t_0 + \delta_n[k_0] + \tau) = u_c(t_0 + \delta_n[k_0] + \tau) \text{ for } \tau \in [0, T_p - \delta_n[k_0]] \tag{93}$$

where $u_c(t_0 + \delta_n[k_0] + \tau)$ is continuously differentiable. Moreover, based on the estimated state vector $\hat{x}_n(t_0)$, we can predict the state vector at time $t_0 + \delta_n[k_0]$ as

$$\hat{x}_n(t_0 + \delta_n[k_0]) = e^{A(\tau)}\hat{x}_n(t_0) + C \int_0^{\delta_n[k_0]} e^{A(\delta_n[k_0]-s)} B \, ds \mu_n[k_0 - 1] \tag{94}$$

According to the above analysis, we have the continuous MPC design within $[t_0 + \delta_n[k_0], t_0 + T_p]$ as follows

$$\text{Given } \hat{x}_n(t_0 + \delta_n[k_0]), \quad \min_{u_c(t_0+\delta_n[k_0]+\tau)} \quad J(u_c(t_0 + \delta_n[k_0] + \tau)) \tag{95}$$

s.t.

(1)

$$u_c(t_0 + \delta_n[k_0] + \tau) \in \mathcal{U}, \;\; x_n(t_0 + \tau) \in \mathcal{X}, \tag{95.a}$$

(2)

$$\begin{aligned} \dot{x}_n(t) &= Ax_n(t) + Bu_c(t) \\ y_n(t) &= Cx_n(t) \end{aligned} \tag{95.b}$$

Using the standard MPC method for linear time-invariant continuous system discussed in [86], we can find an optimal continuous control signal denoted as $u_c^*(t_0 + \delta_n[k_0] + \tau)$ for $\tau \in [0, T_p - \delta_n[k_0]]$, such that $J(u_c(t_0 + \delta_n[k_0] + \tau))$ is minimized.

4.6.2 Discretization

In the second step, we will descretize $u_c^*(t_0 + \delta_n[k_0] + \tau)$ according to the timing constraints in Equation (91.c, (91.d, and (91.e).

Based on the optimal continuous control signal $u_c^*(t_0 + \delta_n[k_0] + \tau)$, we have an optimal plant output as

$$y_n^*(t_0 + \delta_n[k_0] + \tau) = Ce^{A\tau}\hat{x}_n(t_0 + \delta_n[k_0]) + C\int_0^\tau e^{A(\tau - s)}Bu_c^*(t_0 + \delta_n[k_0] + s)\, ds \tag{96}$$

where the plant model in Equation (91.b) is applied. The receding horizon principle in MPC only implements the first part of the calculated future control and the rest are ignored. According to equation (91.c), we know the first part of $u_n(t_0 + \delta_n[k_0] + \tau)$ is

$$u_n(t_0 + \delta_n[k_0] + \tau) = \mu_n[k_0] \text{ for } \tau \in [\, 0, \alpha_n[k_0 + 1] + \delta_n[k_0 + 1] - (\alpha_n[k_0] + \delta_n[k_0])\,) \tag{97}$$

Hence, the discretization step is only concerned with the optimal value of $\mu_n[k_0]$.

To find the optimal value of $\mu_n[k_0]$, we want the plant output $y_n(t_0 + \delta_n[k_0] + \tau)$ under $\mu_n[k_0]$ to track $y_n^*(t_0 + \delta_n[k_0] + \tau)$ as close as possible. Therefore, we define a cost function V as a quadratic function of errors between $y_n^*(t_0 + \delta_n[k_0] + \tau)$ and $y_n(t_0 + \delta_n[k_0] + \tau)$ for $\tau \in [0, \alpha_n[k_0 + 1] + \delta_n[k_0 + 1] - (\alpha_n[k_0] + \delta_n[k_0]))$, i.e.

$$\begin{aligned} V(\mu_n[k_0]) = \int_0^{\alpha_n[k_0+1]+\delta_n[k_0+1]-(\alpha_n[k_0]+\delta_n[k_0])} & (y_n^*(t_0+\delta_n[k_0]+\tau) - y_n(t_0+\delta_n[k_0]+\tau))^T \\ & (y_n^*(t_0+\delta_n[k_0]+\tau) - y_n(t_0+\delta_n[k_0]+\tau))\, d\tau \end{aligned} \tag{98}$$

According to Equation (91.b) and (97), we can express $y_n(t_0 + \delta_n[k_0] + \tau)$ as

$$y_n(t_0 + \delta_n[k_0] + \tau) = Ce^{A(\tau)}\hat{x}_n(t_0 + \delta_n[k_0]) + C\phi(\tau)^T\mu_n[k_0] \tag{99}$$

where

$$\phi(\tau)^T = \int_0^\tau e^{A(\tau-s)}B\,ds$$

Moreover, for notational simplicity, we introduce

$$\begin{aligned}\Psi=&\int_0^{\alpha_n[k_0+1]+\delta_n[k_0+1]-(\alpha_n[k_0]+\delta_n[k_0])}\phi(\tau)C^TCe^{A\tau}\hat{x}_n(t_0+\delta_n[k_0])-\phi(\tau)C^Ty_n^*(t_0+\delta_n[k_0]+\tau)d\tau\\ \Omega=&\int_0^{\alpha_n[k_0+1]+\delta_n[k_0+1]-(\alpha_n[k_0]+\delta_n[k_0])}\phi(\tau)C^TC\phi(\tau)^T\,d\tau\end{aligned} \tag{100}$$

Substituting Equation (99) and (100) into Equation (98), we have simplified the cost function $V(\mu_n[k_0])$ as

$$V(\mu_n[k_0]) = 2\,\mu_n[k_0]^T\,\Psi + \mu_n[k_0]^T\,\Omega\,\mu_n[k_0] + \text{Constant} \tag{101}$$

where the constant term on the right hand side is

$$\begin{aligned}\text{Constant} = &\int_0^{\alpha_n[k_0+1]+\delta_n[k_0+1]-(\alpha_n[k_0]+\delta_n[k_0])}y_n^*(t_0+\delta_n[k_0]+\tau)^Ty_n^*(t_0+\delta_n[k_0]+\tau)\\ &-2\hat{x}_n(t_0+\delta_n[k_0])^Te^{A^T\tau}C^T\left[y_n^*(t_0+\delta_n[k_0]+\tau)+Ce^{A\tau}\hat{x}_n(t_0+\delta_n[k_0])\right]d\tau\end{aligned} \tag{102}$$

Finally, we can see from Equation (101) that minimizing $V(\mu_n[k_0])$ is equivalent to

$$\min_{\mu_n[k_0]} \quad 2\,\mu_n[k_0]^T\,\Psi + \mu_n[k_0]^T\,\Omega\,\mu_n[k_0] \tag{103}$$

Consider the constraints on the control signal, we have the following optimization problem

$$\min_{\mu_n[k_0]} \quad 2\,\mu_n[k_0]^T\,\Psi + \mu_u[k_0]^T\,\Omega\,\mu_u[k_0] \tag{104}$$

subject to

$$\begin{bmatrix} I \\ -I \end{bmatrix}\mu_n[k_0] \le \begin{bmatrix} u_{\max} \\ -u_{\min} \end{bmatrix} \tag{104.a}$$

in which I is an identity matrix.

The objective function (104) is convex and the constraint (104.a) is linear, therefore the optimization problem is convex. It can be solved with efficient Linear Programming (LP) or Quadratic Programming (QP) solvers [14].

4.7 Numeric Simulation

In this section, we use numeric simulations to show that the MPC design based on the hybrid timing model of the CAN bus is effective.

4.7.1 Simulation Setup

We establish an simulation environment for the CAN-based control system according to the system illustrated in Figure 8.

First, the underlying CAN bus in Figure 8 is simulated by Truetime (Version 2.0) [20]. TrueTime is a Matlab/Simulink-based simulator for real-time control system. It has been developed by researchers in Lund University for over 10 years. Truetime provides a network block that support the protocol of the CAN bus. Therefore, we connect nodes to the CAN bus block of the Truetime for the simulation purpose.

Second, we design three feedback control loops sharing the CAN bus. The plant in each feedback loop is an inverted pendulum represented as follows

$$\dot{x}_n(t) = \begin{bmatrix} 0 & 1 \\ a_n & b_n \end{bmatrix} x_n(t) + \begin{bmatrix} 0 \\ c_n \end{bmatrix} u(t) \qquad y_n(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_n(t) \tag{105}$$

The inverted pendulums in three feedback control loops have different coefficients as $[a_1, a_2, a_3] = [98, 65, 44]$, $[b_1, b_2, b_3] = [120, 52, 30]$ and $[c_1, c_2, c_3] = [20, 13, 10]$. The sensor nodes sample the state of the plants at the time interval of 5.0 ms, 7.5 ms, and 10 ms. Each sensor node needs 0.25 ms to process the sampling information and generate a sensor message. The MPC controller node in each feedback control loop computes an optimal control signal $u_n(t)$ that makes the plant output $y_n(t)$ track a

given reference trajectory $\gamma_n(t)$ as close as possible, under the constraint that $-4 \leq u_n(t) \leq 4$. The computation time of MPC is 0.5 ms. The actuator node takes action as soon as the control message is received from the CAN bus. We assume that sensor and control messages have the transmission duration of 0.75 ms and they are assigned unique identifier fields such that $P_1^1[k] < P_1^2[k] < P_2^1[k] < P_2^2[k] < P_3^1[k] < P_3^2[k]$.

Finally, based on the above description, we know that three message chains are being transmitted on the CAN bus. Each message chain has the following characteristics

$$\begin{aligned}
\big[T_1(t), I_1^1(t), C_1^1(t), I_1^2(t), C_1^2(t)\big] &= [5.0, 0.25, 0.75, 0.5, 0.75]\,\text{ms} \\
\big[T_2(t), I_2^1(t), C_2^1(t), I_2^2(t), C_2^2(t)\big] &= [7.5, 0.25, 0.75, 0.5, 0.75]\,\text{ms} \\
\big[T_3(t), I_3^1(t), C_3^1(t), I_3^2(t), C_3^2(t)\big] &= [10\,, 0.25, 0.75, 0.5, 0.75]\,\text{ms}
\end{aligned} \tag{106}$$

4.7.2 Verification of Hybrid Timing Model

Before using the hybrid timing model of the CAN bus for the co-design, we first need to verify the correctness of this model. For this purpose, we compare the delays predicted through the hybrid timing model with the delay observed from the simulation results of Truetime.

Suppose the message chains in Equation (106) are being transmitted on the CAN bus. Figure 12 shows the Truetime simulation of the CAN bus. Table 2 shows the delays $\delta_n[k]$ predicted through the hybrid timing model in Equation (89) and (90). In Figure 12, the value "0.5" indicates that the message is ready for transmission but blocked by other messages on the CAN bus, the value "1" indicates that the message is being transmitted on the CAN bus, and the value "0" indicates that the message finishes transmission.

For illustration, we examine the delay $\delta_3[k]$ in the third feedback control loop. The delays in other feedback control loops can be studied using the exactly same procedure. We know that $\delta_3[k]$ is a time interval between the moment when the sensor

Table 2: Delays Predicted through the hybrid timing model

$\delta_n[k]$	k=1	k=2	k=3	k=4
n=1	2.5 ms	2.25 ms	2.5 ms	2.5 ms
n=2	3.25 ms	2.25 ms	3.25 ms	2.75 ms
n=3	5.25 ms	3.25 ms	3.25 ms	5.25 ms

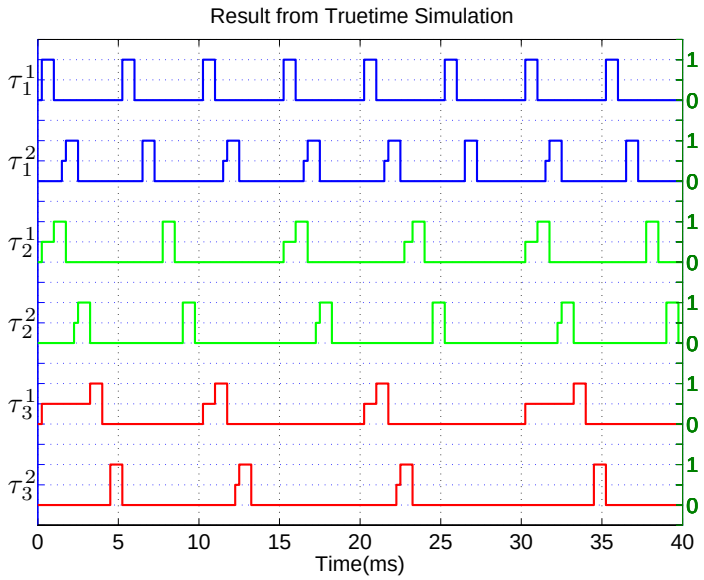


Figure 12: Network Traffics on the CAN bus produced by Truetime Simulation

take measurements and the moment when the actuator take actions. According to Section 4.7.1, the sensor in the third feedback control loops take measurements at 0 ms, 10 ms, 20 ms, and 30 ms. Moreover, by closely examining Figure 12, we observe that the control message τ_3^2 in the third feedback control loop finishes transmission at 5.25 ms, 13.25 ms, 23 ms, and 35.25 ms. Therefore, the observation of Figure 12 shows that the value of $\delta_3[k]$ is 5.25 ms, 3.25 ms, 3 ms, and 5.25 ms, for $1 \leq k \leq 4$. This observation exactly matches the value of $\delta_3[k]$ listed in Table 2. Similarly, we can see that the values of $\delta_1[k]$ and $\delta_2[k]$ observed from Figure 12 also match that listed in Table 2. Therefore, we can claim that the hybrid timing model can accurately describe the message scheduling of the CAN bus.

4.7.3 Improved MPC Design

To thoroughly examine the co-design approach, we study the case, where the CAN-based control system operates in dynamic, uncertain environment. As a result, the messages on the CAN bus may frequently change at runtime. We consider two types of messages adjustments on the CAN bus within the time interval $[1, 1.5]$s. One is the adjustment of the message period as

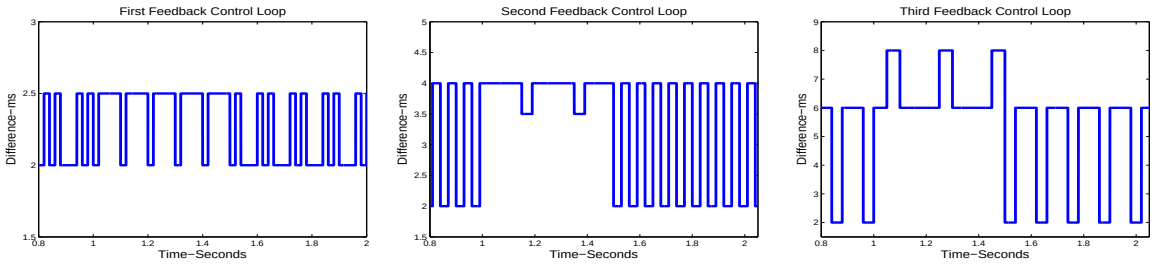
$$[\ T_1(t),\ T_2(t),\ T_3(t)\ ] = [5, 20, 12.5]\ \text{ms} \tag{107}$$

The other type of adjustments is the activation of two sporadic messages on the CAN bus, which has the following characteristics

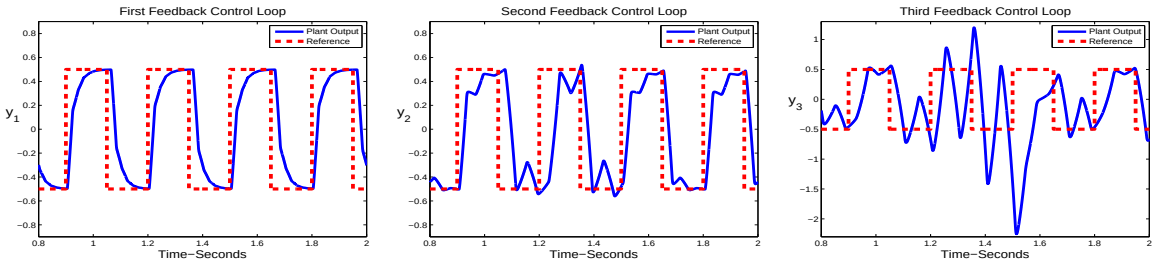
$$\begin{aligned}\left[T_4(t), I_4^1(t), C_4^1(t), I_4^2(t), C_4^2(t)\right] &= [10, 0.05, 0.25, 0, 0]\ \text{ms} \\ \left[T_5(t), I_5^1(t), C_5^1(t), I_5^2(t), C_5^2(t)\right] &= [15, 0.05, 0.25, 0, 0]\ \text{ms}\end{aligned} \tag{108}$$

The sporadic messages are assigned unique identifier field such that $P_5[k] < P_4[k] < P_1^1[k]$. Note that since these adjustments happen at runtime, their characteristics are not available off-line.

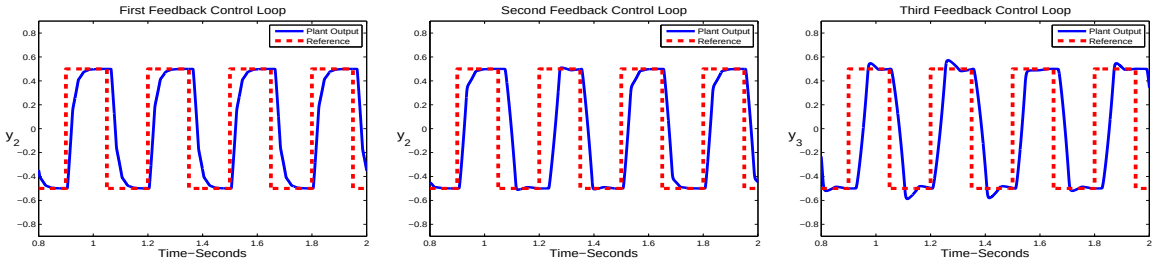
We compare two different approaches of designing MPC for the CAN-based control system. The two approaches differ in their way of predicting response time. In the first approach, the response time $\delta_n[k]$ is predicted off-line through the worst-case analysis discussed in [21, 79, 82]. In second approach, the response time $\delta_n[k]$ is predicted online through the hybrid timing model of the CAN bus. Figure 13 shows the MPC performance of three feedback control loops under the above two different MPC design approaches. The solid line represents the plant output $y_n(t)$ and the dashed line represents the reference trajectory $\gamma_n(t)$. Figure 13.(a) is the difference between actual delays and worst-case response time, Figure 13.(b) shows the results of the the first design approach that uses the worst-case response time, and Figure 13.(c) shows the results of the second design approach that uses the hybrid timing model. We



(a) Difference between Actual Delay and Worst Case Response Time



(b) First MPC Design Approach



(c) Second MPC Design Approaches

Figure 13: MPC performance under different design approaches

obviously see that the second approach (right pictures) gives better performance than the first approach (left pictures). This is because in the second approach, delays are predicted online using the hybrid timing model of the CAN bus, which can accurately predict delay and dynamically account for communication adjustments at run-time.

By comparing Figure 13.(b) with Figure 13.(a), we can see that the larger the difference between actual and worst-case delays is, the worse the control performance is. This is because using the worst-case delay will lead the conservative MPC design, whose performance degrades as the actual delay deviates from the worst-case delay. Finally, the control performance in Figure 13.(c) is good in all situations because the MPC design is based on the actual delays.

Also, it is worth noting that even in the first approach, MPC performance of the first feedback control loop is better than that of the other two loops. This is because the messages in the first feedback control loop are assigned the highest priorities among all messages on the CAN bus, as discussed in Section 4.7.1. As show in Figure 13.(a), the difference between the actual delay and the worst-case response time is small in the first feedback control loop than the other two loops. Therefore, using even the worst-case response time for MPC design in the first feedback control loop can still give out the acceptable performance. However, as such difference increases in the second and third feedback control loops, MPC performance degrades severely.

CHAPTER V

ENERGY MANAGEMENT IN MICRO-GRIDS

Real-time energy management plays an central role in the electric grid. The primary goal of the real-time energy management is to schedule the operations of electrical loads in response to energy supply. The real-time energy management of electrical loads shares some similarities with scheduling tasks on real-time computing systems and transmitting messages over the CAN bus. However, they have also significant differences in the following aspects. First, in both real-time computing system and CAN bus, the available resources, i.e. the number of processors and the bandwidth of the communication channel, are fixed and constant over the entire period of operation, but in the electric grid, the available energy supply may vary with respect to time. Second, at any time, both the processor and the communication network devote all their resource to a single operation, but multiple electrical loads may be scheduled concurrently at the same time, and all loads receive its own demand of power. Finally, tasks on the processor are preemptive while messages on CAN are non-preemptive, but electrical loads can be either preemptive or non-preemptive depending on their functionalities. Because of the above difference, we need to extend our previous results to study the real-time energy management in the electric grid. One important issue in the real-time energy management is feasibility analysis. The goal of feasibility analysis is to check whether a set of electrical loads scheduled by a real-time energy management can achieve user requirements [77]. In recent years, a number of work has been reported in the feasibility analysis of real-time energy management. Facchinetti et. al analyzed a type of electrical loads that control physical process [25]. The feasibility analysis in [25] checks whether these electrical loads can be scheduled by a

real-time energy managment so that specific constraints on the physical process are satisfied. Nghiem et.al. studied the feasibility of scheduling electrical loads under a given constrained peak power [64] [65]. This result is further extended in [48]. It proved that the feasibility relies on the initial condition of electrical loads.

In this chapter, we will study a smaller, self-contained electric grid called micro-grids. Micro-grids have their local sources of power that often include renewable energy such as sunlight, wind, tides, and waves [42]. Utilizing renewable energy in micro-grids has three major advantages. First, it is sustainable and will never run out. Second, it has minimal impact on the environment because it produces little or no waste products such as carbon dioxide or other chemical pollutants. Finally, renewable energy generators generally require less maintenance than traditional ones because their fuel is derived from natural and available resources which reduces the costs of operation. Despite these advantages, renewable energy has presented a challenge of consistent supply rate because it often relies on the environment for its source of power. For example, wind turbines need wind to turn the blades, and solar panels need sunshine to collect heat and make electricity. When these resources are unavailable so is the energy generated from them. Figure 14 shows the power generation of one wind turbine from the Alberta Electric System Operator on July 12th, 2011. The maximum wind power generation at 10:00 is twice as much as the minimum wind power generation at 6:00. As it shows, the generation of the renewable energy is highly variable and dependent on the available resources.

With the integration of renewable energy, micro-grids are able to completely isolate themselves from the national electric grid, and function as a stand alone grid to improve efficiency and local reliability. The goal of the feasibility analysis here is to check whether real-time energy management can schedule electrical loads to guarantee the independent operation of micro-grids under fluctuating renewable energy supply. In case the independent operation of the micro-grid is not possible, our feasibility

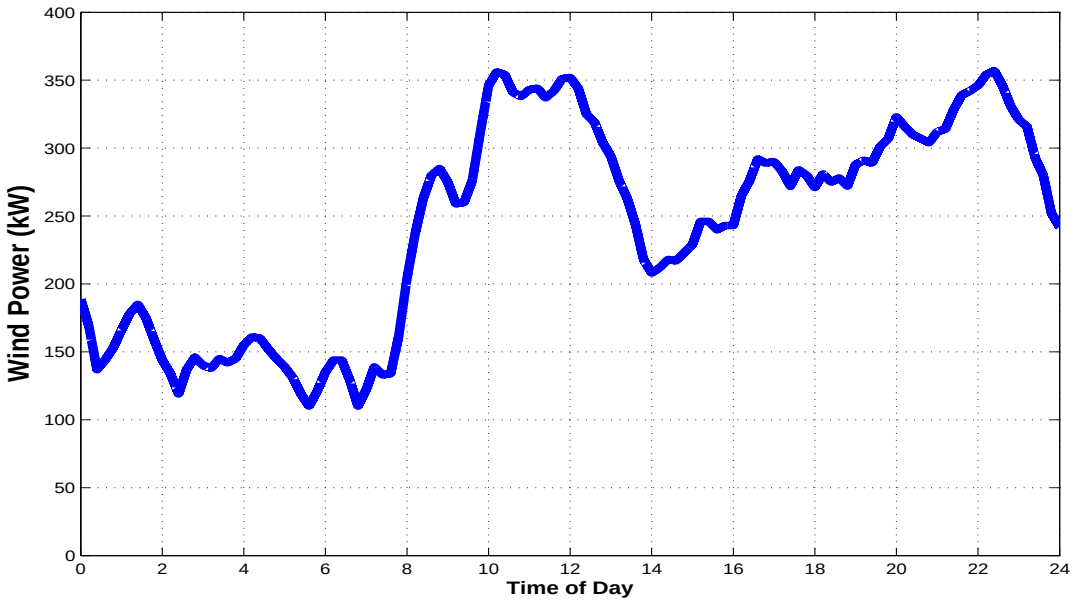


Figure 14: Wind Power Generation reported by Alberta Electric System Operator

gives out accurate predictions regarding when and how much power is insufficient for the independent operation of the micro-grid. Such information allows the operators of micro-grids to take necessary and preventive measures in advance. To the best of our knowledge, this result has not been found in the literature reviewed.

5.1 Introduction of Micro-grids

Micro-grids are modern, small-scale versions of the national electric grid. It consist of electrical loads, on-site generations, and energy storage [42]. On-site generations are local power plants that employ small sources of energy to generate electricity at or near the point of use. Typical sources of energy include fossil fuel (coal, natural gas), and renewable energy (wind, solar) [39]. In contrast to large-scale central generator in the national electric grid, the on-site generation in the micro-grid has little reliance on the distribution and transmission grid because the electricity is generated very near where it is used, perhaps even in the same building. Hence, using on-site generations significantly reduces the amount of energy lost in transmitting electricity, and the number of power lines that must be constructed. Energy storage is a device that

saves the on-site generations for later use. Typical forms of energy storage includes batteries, super-capacitors, and flywheels. Energy storage is helpful in micro-grid applications where the on-site generation and demand of electrical loads cannot be exactly matched all the time. The energy is stored whenever the supply from the on-site generation exceeds the load demand. On the other hand, the power from energy storage is needed whenever the on-site generation are insufficient to supply the load demand. The use of energy storage provides a bridge to balance the supply and demand in micrgorids.

Implementation and integration of on-site power generation and energy storage in micro-grids can bring many benefits. It reduces the risk of a catastrophic power loss and lowers the cost of regular operation. Also, it allows micro-grids to operate independently from the national electric grid, which provide more security against terrorism, natural disasters and other dangers [45]. Moreover, the use of renewable energy for the on-site generation will significantly reduce the carbon footprint on the environment. These benefits have greatly stimulated the adoption of micro-grids in various applications. For example, the military bases are actively deploying micro-grids in order to assure reliable power supply without relying on the national power system [35]. Educational institutions have extensively built micro-grids in their campuses. According to news center of the University of California, San Diego, 90 percent of its annual electricity generation comes from micro-grids. As discussed in [72], micro-grids have become the best solution to isolated, stand-alone areas that may never be connected to the national electric grid due to their remoteness.

5.1.1 Infrastructure of Micro-grids

We consider a typical micro-grid as shown in Figure 15 [42]. The purpose of this micro-grid is to provide energy continuity and security to the small-modular residential or commercial units.

The left hand side of Figure 15 shows the on-site power supply of the micro-grid. The on-site power supply comes from both the renewable energy and fossil fuel generators. Renewable energy includes solar and wind power. They are clean and low cost, but the generation is highly variable as shown in Figure 14. The fossil fuel generator produces constant electricity power all the time.

The right hand side of Figure 15 shows electrical loads and battery banks. Batteries are electrochemical devices that store energy from the on-site generation for later use. The power from the battery is needed whenever the on-site power supply are insufficient to satisfy the demand of all electrical loads. On the other hand, the battery will store the energy whenever the supply from the on-site generation exceeds the load demand.

The arrows in Figure 15 shows the flow of energy. As we can see, the energy flows from the supply to the micro-grid and then from the micro-grid to the electrical loads. Note that the battery bank in the micro-grid can store and supply energy in different situations. Hence, the energy flows both from and to the battery bank.

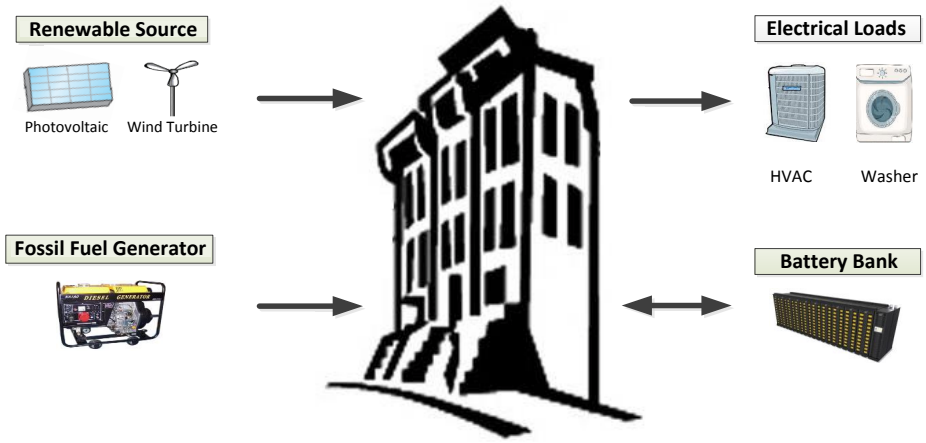


Figure 15: An example of Micro grid

5.1.2 Independent Operation

A fully evolved micro-grid must have the ability to operate independently from the main electric grid for an extended period of time. Successful independent operation

of micro-grids requires that the energy supply from the on-site generation and the battery storage should meet the demand of electrical loads. This requirement can be easily satisfied if all electrical loads in micro-grids are deferrable for a few minutes or hours at little or no cost. A proper real-time energy management can selectively activate/deactivate deferrable loads in response to the energy supply in the micro-grid.

However, in real applications, some electrical loads in the micro-grid may become non-deferrable during the process of operation. We say an electrical load is non-deferrable if (1) the electrical load is non-preemptive and it is currently in the middle of execution; and (2) the electrical load cannot complete its execution before the deadline if not executed immediately. The independent operation of micro-grids at any time t requires that the energy supply from the on-site generation and battery storage should be at least more than the total demand of all non-deferrable electrical loads. Identifying non-deferrable electrical loads at any time t is not easy because an electrical load can switch between deferrable and non-deferrable state during its operation. For example, a non-preemptive electrical load is deferrable before it starts execution, but becomes non-deferrable in the middle of operation; and a preemptive electrical loads is deferrable at the beginning, but may become non-deferrable if the further delay of execution will make electrical loads miss their deadlines.

As discussed above, we know that whether an electrical load is deferrable or non-deferrable depends on its current status of execution. In the following sections, we will establish a mathematical model that describes the status evolution of electrical loads in the micro-grids. Based on this model, we can identify all non-deferrable electrical loads at any time t, which will facilitate the feasibility analysis of real-time energy management.

5.2 Models of Micro-grids

In this section, we develop a set of models capable of describing different components in the micro-grids. Base on these models, we can then analyze the dynamic behaviors of micro-grids in the next section.

5.2.1 Electrical Loads

Without loss of generality, we assume that the micro-grid contains a set of electrical loads $\Gamma = \{\tau_1, \cdots, \tau_N\}$. Each electrical load τ_n in Γ consists of a sequence of instances, and each instance corresponds to one operation request of τ_n. We use $\tau_n[k]$ to denote the k-th instance of τ_n. $\tau_n[k]$ can be characterized by the requested time $\alpha_n[k]$, inter-request time interval $T_n[k]$ between $\alpha_n[k+1]$ and $\alpha_n[k]$, relative deadline $D_n[k]$, operational power $E_n[k]$, operation time $C_n[k]$, preemption $F_n[k]$, and priority $P_n[k]$. Note $F_n[k] = 0$ denotes that τ_n is non-preemptive during operation and $F_n[k] = 1$ denotes that τ_n is preemptive during operation. The smaller value of $P_n[k]$ denotes a higher priority. In the following part of this section, we will study the representation of different types of electrical loads using the above notations.

First, we give an example of a simple electrical loads such as rice cookers. We assume that a rice cooker has the specifications as follows.

Example 5.2.1 *A rice cooker operates once every 24 hours and each operation will take one hour. The operation is requested at 9:00am and must finish before 7:00pm. The operational power is* $310w$*. The operation is non-preemptive once started. The rice cooker has the second highest priority.*

Based on the above requirement, we can characterize the rice cooker with the following parameters:

$$\alpha_n[k] = 24k+9,\ T_n[k] = 24,\ D_n[k] = 10,\ E_n[k] = 310,\ C_n[k] = 1,\ F_n[k] = 0,\ P_n[k] = 2 \tag{109}$$

in which $F_n[k] = 0$ denotes that the electrical load is non-preemptive during its operation. On the other hand, $F_n[k] = 1$ if an electrical load is preemptive during its operation.

Second, we introduce electrical loads with multiple internal operation phases. For example, the operation of dish washers go through five phases as: pre-wash, wash, first rinse, drain, second rinse, and dry. The five operation phases must follow a strict sequential order since the next operation phase only starts after its preceding operation phase is completed. Also, each operation phase may have different execution time, power and preemption property. Table 3 shows a detailed specification of a dishwasher. According to the specification, we can express the dishwasher as

$$\begin{aligned} E_n[k] &= [\,64.20,\ 1517.8,\ 103.8,\ 8.2,\ 1872.3,\ 1.9\,] \\ C_n[k] &= [\ 14.9\,,\ \ 32.1\ ,\ 10.1\,,\ 4.3,\ \ 18.3\ ,\ 51.4\,] \\ F_n[k] &= [\ \ 1\ \ ,\ \ \ 1\ \ \ ,\ \ 0\ \ ,\ 1\,,\ \ \ 1\ \ ,\ \ 1\ \ ] \end{aligned}$$

where $E_n[k]$, $C_n[k]$ and $F_n[k]$ are vectors with multiple elements, and each element in the vector corresponds to one operation phase of the dishwasher.

Table 3: Dishwasher Specification

Phase	Pre-wash	Wash	1st Rinse	Drain	2nd Rinse	Dry
Operation Power	64.20w	1517.8w	103.8w	8.2w	1872.3w	1.9w
Operation Time	14.9m	32.1m	10.1m	4.3m	18.3m	51.4m
Preemption	Yes	Yes	No	Yes	No	Yes

Third, we introduce electrical loads subject to the precedence constraints with other electrical loads. For instance, a dryer machine cannot start its operation until a washing machine has completed its operation. To model the precedence constraints among multiple electrical loads, we introduce a new concept called comprehensive electrical loads that is formally defined as follow.

Definition 5.2.2 *A comprehensive electrical load is composed of several distinct parts that are linked in a strict order. Each part corresponds to one electrical load.*

With the comprehensive electrical loads well defined, we can view any group of precedence constrained electrical load as comprehensive electrical loads. For any comprehensive electrical load, $\{E_n[k], C_n[k], F_n[k], P_n[k]\}$ are vectors containing the characteristics of each electrical load. Therefore, the mathematical expression of comprehensive electrical loads is similar to electrical loads with internal operation phases. Note that $P_n[k]$ is a vector for comprehensive electrical loads as each individual load has a different priority.

Finally, we introduce electrical loads with dynamics changing according to the physical environment. Meliopoulos et.al. [57] proposed mathematical models that represents thermostatically controlled loads in a residential home. Consider air conditioners (AC) as an example. We use x to denote the house temperature inside the house, and TP_{out} denote the outside temperature. According to the dynamic model in [57], we can represent the operation of AC as

$$\dot{x} = -\frac{G_{\text{out}}}{C_h}(x - TP_{\text{out}}) + \frac{1}{C_h} n_{\text{ac}} P_{ac} u \tag{110}$$

where G_{out} is the thermal conductance between house and outside environment, C_h the thermal capacitance of the house, n_{ac} the coefficient of performance of AC, P_{ac} the power of AC, and u is the duty cycle of AC. Note that AC will cycle on and off periodically. Therefore, the duty cycle of AC is the percentage of one period in which AC is on. By controlling the duty cycle u, AC will guarantee that the house temperature will stay within a bounded range such that $TP_{\min} \leq x \leq TP_{\max}$. Suppose that x is currently at a stable point x_{stable} such that $TP_{\min} \leq x_{\text{stable}} \leq TP_{\max}$. To guarantee that x always stays at this point, we must have that $\dot{x} = 0$, i.e.

$$0 = -\frac{G_{\text{out}}}{C_h}(x_{\text{stable}} - TP_{\text{out}}) + \frac{1}{C_h} n_{\text{ac}} P_{ac} u \tag{111}$$

which implies that

$$u = \frac{G_{\text{out}}}{n_{\text{ac}} P_{\text{ac}}}(x_{\text{stable}} - TP_{\text{out}}) \tag{112}$$

Given the duty cycle u, we have the execution time of AC as

$$C_n[k] = T_n[k]\, u = T_n[k]\, \frac{G_{\text{out}}}{n_{\text{ac}}\, P_{\text{ac}}}(x_{\text{stable}} - TP_{\text{out}}) \tag{113}$$

where $T_n[k]$ is the period of one on and off cycle. As it shows, the execution time of AC will dynamically change according to the outside temperature TP_{out}.

According the discussions, we can represent different types of electrical loads as the tuple $\{\, C_n[k]\,,\, E_n[k]\,,\, D_n[k]\,,\, T_n[k]\,,\, F_n[k]\,, P_n[k]\}$. Since we want to model the dynamics of the real-time energy management at any time t, we define the characteristics of electrical loads in continuous time domain as follows

Definition 5.2.3 *At any time t, an instance of τ_n is effective if and only if it is the nearest instance released before or at time t, i.e. $\tau_n[k]$ is effective at time t if and only if*

$$\alpha_n[k] \leq t < \alpha_n[k+1]. \tag{114}$$

Definition 5.2.4 *At any time t, $C_n(t)$, $E_n(t)$, $D_n(t)$, $T_n(t)$, $F_n(t)$ and $P_n(t)$ are respectively defined as the operation time, power, relative deadline, inter-request time, preemption, and the priority of the effective instance of τ_n, i.e.*

$$\text{if } \alpha_n[k] \leq t < \alpha_n[k+1]$$
$$C_n(t){=}C_n[k],\ E_n(t){=}E_n[k],\ D_n(t){=}D_n[k],\ T_n(t){=}T_n[k],\ F_n(t){=}F_n[k],\ P_n(t){=}P_n[k] \tag{115}$$

Therefore, electrical loads in the micro grid can be represented in continuous time domain as $\{\, C_n(t)\,,\, E_n(t)\,,\, D_n(t)\,,\, T_n(t)\,,\, F_n(t)\,, P_n(t)\}_{n=1}^{N}$.

5.2.2 On-site Generation and Battery Bank

In micro-grids, a noticeable portion of electricity is generated on-site from different sources of energy. We formally define on-site generation of electricity as follows

Definition 5.2.5 *At any time t,* $\text{EG}(t)$ *is defined as the on-site electricity generation in a micro-grid.*

$\mathrm{EG}(t)$ includes the electricity from both the fossil fuel generator and the renewable energy. Since the electricity generated from renewable energy is highly variable as shown in Figure 14, the total on-site electricity generation $\mathrm{EG}(t)$ changes with respect to time.

Battery bank is used in micro-grid applications where the generation and load demand cannot be exactly matched. It increases the stability and reliability of the micro-grids. We formally define the battery bank as follows

Definition 5.2.6 *At any time t, $\mathrm{SOC}(t)$ is defined as the state of charge of the battery bank in the micro-grid. $\mathcal{B}_{\mathrm{power}}$ is defined as maximum charge/discharge rate of the battery bank. $\mathcal{B}_{\mathrm{capacity}}$ is defined as capacity of the battery bank.*

$\mathrm{SOC}(t)$ indicates the percentage of energy remaining in the battery bank. Ideally speaking, a fully charged and deeply discharged battery has $\mathrm{SOC}(t) = 100\%$ and $\mathrm{SOC}(t) = 0\%$, respectively. However, in real applications, batteries should not be discharged below 20% of its SOC. Otherwise, the battery life will be significantly shortened. In order to protect the battery bank, we put the following constraints on the battery operation such that

$$20\% \leq \mathrm{SOC}(t) \leq 100\% \tag{116}$$

According to the above constraint, we know that the maximum power output of the battery bank at any time t depends on its state of charge. When the state of charge is larger than 20%, we have the maximum output power of the battery bank as $\mathcal{B}_{\mathrm{power}}$, i.e.

$$\text{when } \mathrm{SOC}(t) > 20\%, \quad \text{maximum power output of the battery bank is } \mathcal{B}_{\mathrm{power}} \tag{117}$$

On the other hand, when the state of charge drops to 20%, the batteries is not allowed to output any power, i.e.

$$\text{when } \mathrm{SOC}(t) = 20\%, \quad \text{maximum power output of the battery bank is } 0 \tag{118}$$

According to Equation (117) and (118), the maximum power output of the battery bank at any time t is expressed as

$$\text{sgn}(\text{SOC}(t) - 20\%)\mathcal{B}_{\text{power}} \tag{119}$$

where $\text{sgn}(x)$ is a signum function such that $\text{sgn}(x) = 1$ if $x > 0$, $\text{sgn}(x) = 0$ if $x = 0$, and $\text{sgn}(x) = -1$ if $x < 0$.

Remark 5.2.7 *We measure the capacity of the battery bank $\mathcal{B}_{\text{capacity}}$ in kilowatt-hours (kWh). Another common measure of the capacity of the battery bank is Ah, defined as the number of hours for which the battery bank can provide a current equal to the discharge rate at the nominal battery voltage. The kWh capacity can be approximated from the Ah capacity by multiplying the Ah capacity with the nominal battery voltage.*

Remark 5.2.8 *In most batteries, discharge/charge rate is expressed as a C-rate. 1 C-rate means that the maximum discharge/charge current will discharge/charge the entire battery in 1 hour. For example, consider a battery has the capacity of 100 Ah. If this battery has 1 C-rate, then the maximum discharge/charge current for this battery is 100A; if the battery has 5 C-rate, the maximum discharge/charge current is 500A, and if the battery has 0.5 C-rate, the maximum discharge/charge current is 50A. $\mathcal{B}_{\text{power}}$ can be approximated from the C-rate current by multiplying the C-rate current with the nominal battery voltage.*

5.3 Dynamic Timing Model

In the previous sections, we have shown that the micro-grid consists of three major components represented as follows

1. On-site electricity generation is represented as $\text{EG}(t)$;

2. Batteries are represented as $\{\text{SOC}(t), \mathcal{B}_{\text{power}}, \mathcal{B}_{\text{capacity}}\}$;

3. Electrical loads $\{ C_n(t) , E_n(t) , D_n(t) , T_n(t) , F_n(t) , P_n(t)\}_{n=1}^{N}$, where N denotes the total number of electrical loads in the micro-grid.

The establishment of these mathematical expressions will allow us to analytically study the dynamics of real-time energy management in the micro grid.

5.3.1 State Vector of Electrical Loads

We consider a micro grid with a set of electricity loads $\{\tau_1, \cdots, \tau_N\}$. To study the progression of electrical loads under the real-time energy management, we introduce a state vector $Z(t) = [S(t), R(t), O(t)]^T$ that describes the current status of $\{\tau_1, \cdots, \tau_N\}$ at any time t.

Definition 5.3.1 *The dynamic inter-request time is defined as $S(t) = [s_1(t), \cdots, s_N(t)]$, where $s_n(t)$, for $n = 1, 2, \cdots, N$, denotes how long after t the next instance of τ_n will be requested.*

Definition 5.3.2 *The residue is defined as $R(t) = [r_1(t), ..., r_N(t)]$, where $r_n(t)$, for $n = 1, 2, ..., N$, denotes the remaining operation time of the current instance of τ_n* after *time t.*

Definition 5.3.3 *The dynamic response time is defined as $O(t) = [o_1(t), ..., o_N(t)]$, where $o_n(t)$, for $n = 1, 2, ..., N$, denotes how much time has elapsed before the current instance of τ_n finishes operation.*

Based on the above definitions, we know that the progression of electrical loads under the real-time energy management can be represented through the evolution of the state vector $Z(t)$. In the following part, we will study the evolution of $Z(t) = [S(t), R(t), O(t)]^T$ under real-time energy management.

5.3.2 Non-Deferrable Electrical Loads

According to Section 5.1.2, we know that two types of electrical loads in the micrgorid are considered as non-deferrable. The first type of non-deferrable electrical loads are electrical loads that could not complete execution before deadline if further delayed. This type of electrical loads is represented as

$$\{\, i \,|\, o_i(t) + r_i(t) = D_i(t) \,\} \tag{120}$$

where $o_i(t)$ denotes the delay of τ_i since its time of request, $r_i(t)$ denotes the remaining operation of τ_i, and $o_i(t) + r_i(t) = D_i(t)$ indicates that τ_i can complete before its deadline if and only if it continuously execute from now on. The second type of non-deferrable electrical loads are electrical loads that are non-preemptive and currently in the middle of operation. This type of electrical loads is represented as

$$\{\, i \,|\, F_i(t) = 0, \text{and } 0 < r_i(t) < C_i(t) \,\} \tag{121}$$

where $F_i(t) = 0$ denotes that τ_i is non-preemptive, and $0 < r_i(t) < C_i(t)$ denotes that τ_i is in the middle of operation. Note that any non-preemptive electrical load τ_i with $r_i(t) = 0$ or $r_i(t) = C_i(t)$ is excluded from the set in Equation (121). This is because (1) $r_i(t) = 0$ denotes that τ_i has completed execution before t and no need to consider electrical loads already completed; and (2) $r_i(t) = C_i(t)$ denotes that τ_i has NOT started execution yet, and non-preemptive electrical loads are deferrable before its start of execution.

According to the above analysis, we can formally define and represent non-deferrable electrical loads as follows

Definition 5.3.4 *We use* NonDefer(t) *to denote a set of non-deferrable electrical loads that must be executed immediately at time* t. NonDefer(t) *can be represented as*

$$\text{NonDefer}(t) = \{\, i \,|\, o_i(t) + r_i(t) = D_i(t)\} \bigcup \{\, i \,|\, F_i(t) = 0, \text{and } 0 < r_i(t) < C_i(t) \,\} \tag{122}$$

5.3.3 Real-time Energy Management

In this section, we rigorously define real-time energy management using mathematical expression. This mathematical expression will be then used in analyzing the dynamics of electrical loads. We let $\{1, \cdots, N\}$ denote the set of indices of total electrical loads in the micro-grid and $\mathrm{OP}(t)$ denote the set of indices of electrical loads executing at time t.

Definition 5.3.5 *A* real-time energy management *is a set-valued map between* R^+ *and the collection of all subsets of* $\{1, \cdots, N\}$. *It is parameterized as* $\mathrm{OP}(t) : R^+ \to 2^{\{1,\cdots,N\}}$, *where* $t \in R^+$.

At any time t, $\mathrm{OP}(t)$ should at least contain non-deferrable electrical loads. If the energy supply is larger than the demand of all non-deferrable electrical loads, the real-time energy management will activate the execution of other deferrable electrical loads in terms of their priorities.

At any time t, we can construct the real-time energy management $\mathrm{OP}(t)$ through three steps as follows

First Step: Initialization. In this step, we initialize $\mathrm{OP}(t)$ as a set of indices of non-deferrable electrical loads at time t, i.e.

$$\mathrm{OP}(t) = \mathrm{NonDefer}(t) \tag{123}$$

Moreover, we initialize an scheduling pool $\mathrm{SCH}(t)$ as a set of deferrable electrical loads that have remaining operation time, i.e.

$$\mathrm{SCH}(t) = \{\, i \,|\, r_i(t) > 0 \,\} - \mathrm{NonDefer}(t), \tag{124}$$

where $\{\, i \,|\, r_i(t) > 0 \,\}$ denotes a set of electrical loads with remaining operation time and "$-$" denotes the substraction between two sets. Note that $\{\, i \,|\, r_i(t) = 0 \,\}$ are excluded from the scheduling pool $\mathrm{SCH}(t)$ because $r_i(t) = 0$ denotes electrical loads that

have completed execution before time t. We do not need to consider the scheduling of electrical loads that have completed execution, as they will not request any energy.

Second Step: Scheduling. In this step, we decide whether the highest priority electrical load in SCH(t) can be scheduled to operate at time t. The highest priority electrical load in SCH(t) can be denoted as τ_n such that $n = \max_{i \in \mathrm{SCH}(t)} P_i(t)$. If τ_n satisfies the following condition,

$$\sum_{i \in OP(t)} E_i(t) + E_n(t) \leq \mathrm{EG}(t) + \mathrm{sgn}(\mathrm{SOC}(t) - 20\%)\mathcal{B}_{\mathrm{power}} \tag{125}$$

where the first element on the left hand side represents the demand of electrical loads that have been scheduled to execute at time t, the second element on the left hand side represents the demand of τ_n, and the right hand side represents the total supply of on-site generation and the battery storage. Equation (125) indicates that the total supply at current time t is enough to satisfy the demand of both electrical loads in OP(t) and the electrical load τ_n. In this case, τ_n will be scheduled to execute at time t, i.e.

$$\mathrm{OP}(t) = \mathrm{OP}(t) + \{\, n \,\} \tag{126}$$

On the other hand, if the electrical load τ_n satisfies the following condition,

$$\sum_{i \in \mathrm{OP}(t)} E_i(t) + E_n(t) > \mathrm{EG}(t) + \mathrm{sgn}(\mathrm{SOC}(t) - 20\%)\mathcal{B}_{\mathrm{power}}. \tag{127}$$

which indicates that the total energy at current time t is NOT enough to satisfy the demand of both electrical loads in OP(t) and the electrical load τ_n. In this case, τ_n will NOT be scheduled to execute at time t, i.e.

$$\mathrm{OP}(t) = \mathrm{OP}(t) \tag{128}$$

Third Step: Update and Check. In this step, we update scheduling pool SCH(t) and check the the number of electrical loads in SCH(t). Since the execution of electrical load τ_n has been decided in the second step, we need to remove τ_n from the

scheduling pool, i.e.

$$\mathrm{SCH}(t) = \mathrm{SCH}(t) - \{\, n \,\} \tag{129}$$

If the scheduling pool is NOT empty, i.e. $\mathrm{SCH}(t) \neq \emptyset$, real-time energy management needs to decide the execution of other electrical loads in $\mathrm{SCH}(t)$. In this case, go back to Step 2. On the other hand, if the scheduling pool is empty, i.e. $\mathrm{SCH}(t) \neq \emptyset$, the execution of all deferrable electrical loads have been decided and the construction of $\mathrm{OP}(t)$ finishes.

Algorithm 3 gives an detailed implementation of real-time energy management. The input of the algorithm are the current status of electrical loads, on-site generation, and battery SOC. The output of the algorithm is a set of electrical loads that are scheduled to execute at time t. In the next section, we will show that at any time t, an electrical load τ_n has different evolution dynamics, depending on whether τ_n is scheduled to execute (i.e. $n \in \mathrm{OP}(t)$) or not (i.e. $n \notin \mathrm{OP}(t)$).

Algorithm 3: Real-time Energy Management

Data: $Z(t)$, $\mathrm{SOC}(t)$, $\mathrm{EG}(t)$, $\mathcal{B}_{\mathrm{power}}$, $\{C_n(t), E_n(t), D_n(t), T_n(t), F_n(t), P_n(t)\}_{n=1}^{N}$
Result: $\mathrm{OP}(t)$

$\mathrm{NonDefer}(t) = \{i | o_i(t) + r_i(t) = D_i(t)\} \cup \{i | F_i(t) = 0, \text{and } 0 < r_i(t) < C_i(t)\}$;
/*1st Step: Initialization*/
$\mathrm{OP}(t) = \mathrm{NonDefer}(t)$;
$\mathrm{SCH}(t) = \{\, i \,|\, r_i(t) > 0 \,\} - \mathrm{NonDefer}(t)$;
while $\mathrm{SCH}(t) \neq \emptyset$ **do**
  /*2nd Step: Scheduling*/
  $n = \max_{i \in \mathrm{SCH}(t)} P_i(t)$;
  if $\sum_{i \in \mathrm{OP}(t)} E_i(t) + E_n(t) \leq \mathrm{EG}(t) + \mathrm{sgn}(\mathrm{SOC}(t) - 20\%)\mathcal{B}_{\mathrm{power}}$ **then**
    $\mathrm{OP}(t) = \mathrm{OP}(t) + \{\, n \,\}$;
  else
    $\mathrm{OP}(t) = \mathrm{OP}(t)$;
  /*3nd Step: Update and Check*/
  $\mathrm{SCH}(t) = \mathrm{SCH}(t) - \{\, n \,\}$;
return $\mathrm{OP}(t)$;

5.3.4 Evolution of Electrical Loads

The state vector $Z(t)$ contains the status for a set of N electrical loads. Therefore, the evolution of $Z(t)$ can be obtained through the evolution of each electrical load. For each electrical load τ_n, its state vector $[s_n(t), r_n(t), o_n(t)]^T$ can evolve in both continuous and discrete ways.

First, we discuss the discrete evolution of $[s_n(t), r_n(t), o_n(t)]^T$. According to Definition 5.3.1 $\sim$ Definition 5.3.3, we know that $[s_n(t), r_n(t), o_n(t)]^T$ represents the status of the current instance of τ_n at time t. Therefore, if a new instance of τ_n arrives at time t, i.e. $s_n(t) = 0$, the state vector will update according to the characteristics of the new instance. Thus, we have that

$$\text{if } s_n(t) = 0 : \quad \begin{bmatrix} s_n(t) \\ r_n(t) \\ o_n(t) \end{bmatrix} = \begin{bmatrix} T_n(t) \\ C_n(t) \\ 0 \end{bmatrix}. \tag{130}$$

Next, we discuss the continuous evolution of $[s_n(t), r_n(t), o_n(t)]^T$. According to Section 5.3.3, the execution of τ_n will continue at time t when $n \in \text{OP}(t)$ and stop at time t when $n \notin \text{OP}(t)$. Therefore, the continuous evolution of $[s_n(t), r_n(t), o_n(t)]^T$ depends on the execution status of τ_n. We use Δt to denote an arbitrary small value.
Case 1: τ_n will execute at time t, i.e. $n \in \text{OP}(t)$. According to Definition 5.3.1 $\sim$ Definition 5.3.3, we know that $s_n(t)$ and $r_n(t)$ will decrease from t to $t+\Delta t$, and $o_n(t)$ will increase from t to $t + \Delta t$, i.e.

$$s_n(t+\Delta t) = s_n(t) - \Delta t, \quad r_n(t+\Delta t) = r_n(t) - \Delta t, \quad o_n(t+\Delta t) = o_n(t) + \Delta t \tag{131}$$

which implies the continuous evolution of $[s_n(t), r_n(t), o_n(t)]^T$ at time t is

$$\text{if } s_n(t) \neq 0 \text{ and } n \in \text{OP}(t) : \quad \begin{bmatrix} \dot{s}_n(t) \\ \dot{r}_n(t) \\ \dot{o}_n(t) \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}. \tag{132}$$

Case 2: τ_n will NOT execute at time t, i.e. $n \notin \text{OP}(t)$. In this case, $s_n(t)$ will continuously decrease from t to $t + \Delta t$, i.e.

$$s_n(t + \Delta t) = s_n(t) - \Delta t \tag{133}$$

Moreover, $r_n(t)$ denotes the residue of the current instance of τ_n. Since the execution of τ_n stops at time t, the residue $r_n(t)$ will NOT decrease in this case because , i.e.

$$r_n(t + \Delta t) = r_n(t) \tag{134}$$

Finally, the evolution of $o_n(t)$ from t to $t + \Delta t$ is a little involved. It depends whether the current instance of τ_n has completed its execution before time t. If the current instance of τ_n has completed execution before time t, i.e. $r_n(t) = 0$, the dynamic response time $o_n(t)$ keeps constant, i.e. $o_n(t + \Delta t) = o_n(t)$. On the other hand, if the current instance of τ_n has NOT completed execution before time t, i.e. $r_n(t) > 0$, the dynamic response time $o_n(t)$ increases, i.e. $o_n(t + \Delta t) = o_n(t) + \Delta t$. Based on the above analysis, we have the continuous evolution of $o_n(t)$ as

$$o_n(t + \Delta t) = o_n(t) + \text{sgn}(r_n(t))\Delta t \tag{135}$$

According to Equation (133) $\sim$ (135), the continuous evolution of $[s_n(t), r_n(t), o_n(t)]^T$ at time t can be expressed as

$$\text{if } s_n(t) \neq 0 \text{ and } n \notin \text{OP}(t): \quad \begin{bmatrix} \dot{s}_n(t) \\ \dot{r}_n(t) \\ \dot{o}_n(t) \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ \text{sgn}(r_n(t)) \end{bmatrix}. \tag{136}$$

5.3.5 Battery State of Charge Evolution

At any time t, the status of the battery bank in the micro grid can be represented by the SOC(t). In this section, we are interested in studying the evolution of SOC(t) with respect to time. As discussed before, the batteries in the micro grid have two modes of operation. In the first mode of operation, the on-site energy is insufficient to

supply load demands and the batteries will provide power to electrical loads. In the second mode of operation, the on-site electricity generation exceeds the load demand and the batteries store extra energy for later use. We will analyze the evolution of $\mathrm{SOC}(t)$ in two different modes of operation, respectively.

First, we study the evolution of $\mathrm{SOC}(t)$ when the batteries are storing energy for later use. This mode of operation happens when the following condition is satisfied

$$\sum_{i \in \mathrm{OP}(t)} E_i(t) < \mathrm{EG}(t). \tag{137}$$

Therefore, the extra power stored in batteries at time t is $\mathrm{EG}(t) - \sum_{i \in \mathrm{OP}(t)} E_i(t)$. If we use Δt to denote an arbitrarily small time. Then, we have that

$$\left(\mathrm{SOC}(t + \Delta t) - \mathrm{SOC}(t)\right) \mathcal{B}_{\mathrm{capacity}} = \left(\mathrm{EG}(t) - \sum_{i \in \mathrm{OP}(t)} E_i(t)\right) \Delta t \tag{138}$$

which implies that

$$\dot{\mathrm{SOC}}(t) = \lim_{\Delta t \to 0} \frac{\mathrm{SOC}(t + \Delta t) - \mathrm{SOC}(t)}{\Delta t} = \frac{\mathrm{EG}(t) - \sum_{i \in \mathrm{OP}(t)} E_i(t)}{\mathcal{B}_{\mathrm{capacity}}} \tag{139}$$

According to Equation (137), we know that $\dot{\mathrm{SOC}}(t) > 0$, which is consistent with the fact that the battery bank is storing energy.

Next, we consider the evolution of $\mathrm{SOC}(t)$ when the batteries are providing energy to electrical loads. This mode of operation happens when the following condition is satisfied

$$\sum_{i \in \mathrm{OP}(t)} E_i(t) > \mathrm{EG}(t). \tag{140}$$

Therefore, the power provided by batteries at time t is $\sum_{i \in \mathrm{OP}(t)} E_i(t) - \mathrm{EG}(t)$. Then, we have that

$$\left(\mathrm{SOC}(t) - \mathrm{SOC}(t + \Delta t)\right) \mathcal{B}_{\mathrm{capacity}} = \left(\sum_{i \in \mathrm{OP}(t)} E_i(t) - \mathrm{EG}(t)\right) \Delta t \tag{141}$$

which implies that

$$\dot{\mathrm{SOC}}(t) = \lim_{\Delta t \to 0} \frac{\mathrm{SOC}(t + \Delta t) - \mathrm{SOC}(t)}{\Delta t} = \frac{\mathrm{EG}(t) - \sum_{i \in \mathrm{OP}(t)} E_i(t)}{\mathcal{B}_{\mathrm{capacity}}} \tag{142}$$

According to Equation (140), we know that $\dot{\text{SOC}}(t) < 0$, which is consistent with the fact that the battery bank is providing energy.

In summary, $\text{SOC}(t)$ has the same expression of evolution in two different modes of battery operations, as shown in Equation (139) and (142).

5.4 Feasibility Analysis

A fully evolved micro-grid should be able to operate independent from the main grid for an extended period of time. As discussed in Section 5.1.2, the independent operation of the micro-grid requires that the energy supply from the on-site generation and the battery storage should meet the demand of all non-deferrable electrical loads. In this section, we will propose a necessary and sufficient feasibility analysis that checks whether the requirement for the independent operation can be satisfied. Based on this feasibility analysis, we can also accurately predict when and how much power is insufficient for the independent operation of micro-grids.

5.4.1 Necessary and Sufficient Feasibility Analysis

As discussed at the beginning of this chapter, our feasibility analysis of real-time energy management checks whether the micro-grids can operate independently from the main electric grid. Here, we propose a necessary and sufficient condition for feasibility analysis as follows

Claim 5.4.1 *A micro-grid can operate independently from the main electric grid within* $[t_a, t_b]$ *if and only if it satisfies the following conditions for all* $t \in [t_a, t_b]$,

$$\sum_{i \in \text{NonDefer}(t)} E_i(t) \leq \text{EG}(t) + \text{sgn}(\text{SOC}(t) - 20\%)\mathcal{B}_{\text{power}}. \tag{143}$$

Proof 5.4.2 *According to Definition 5.3.4 and Equation (122), we know* $\text{NonDefer}(t)$ *denotes a set of non-deferrable electrical loads that must be executed at time* t. *Since* $E_i(t)$ *denotes the power demand of one non-deferrable electrical load* τ_i, *the left hand*

side of Equation (143) denotes the demand of all non-deferrable electrical loads at current time t.

On the other hand, $\mathrm{EG}(t)$ *denotes the on-site generation according to Definition 5.2.5, and* $\mathrm{sgn}(\mathrm{SOC}(t)-20\%)\mathcal{B}_{\mathrm{power}}$ *denotes the maximum power output of the battery storage according to Equation (119). Hence, the right hand side of Equation (143) denotes the total supply from both the on-site generation and battery storage at time t.*

Therefore, the micro-grid can run independently at time t if and only if the left hand side(power demand) is smaller or equal to the right hand side(power supply). The micro-grid can run independently within $[t_a, t_b]$ *if and only if the inequality condition is satisfied for any time* $t \in [t_a, t_b]$.

5.4.2 Deficiency in Power Supply

A successful feasibility analysis in Claim 5.4.1 checks whether the micro-grid can run independently from the main electric grid. Based on this analysis, we can predict when and how much power is insufficient for the independent operation of the micro-grids.

The power deficiency depends on the relation between the power demand and supply.

Case1: the power demand of all non-deferrable electrical loads is smaller or equal to the power supply from the on-site generation and the battery storage, i.e.

$$\sum_{i \in \mathrm{NonDefer}(t)} E_i(t) \leq \mathrm{EG}(t) + \mathrm{sgn}(\mathrm{SOC}(t) - 20\%)\mathcal{B}_{\mathrm{power}}. \tag{144}$$

In this case, the micro-grid can run independently from the main electric grid.

Case2: the power demand of all non-deferrable electrical loads is larger than the power supply from the on-site generation and the battery storage, i.e

$$\sum_{i \in \mathrm{NonDefer}(t)} E_i(t) > \mathrm{EG}(t) + \mathrm{sgn}(\mathrm{SOC}(t) - 20\%)\mathcal{B}_{\mathrm{power}}. \tag{145}$$

In this case, the micro-grid cannot run independently and the amount of insufficient power supply can be expressed as

$$\sum_{i \in \text{NonDefer}(t)} E_i(t) - \text{EG}(t) - \text{sgn}(\text{SOC}(t) - 20\%)\mathcal{B}_{\text{power}} \tag{146}$$

Based on the above analysis, we have the following claim about the insufficient power supply of the micro-grid

Claim 5.4.3 *At any time t, the amount of insufficient power for the independent operation of micro-grid can be expressed as*

$$\max\{\, 0\,, \sum_{i \in \text{NonDefer}(t)} E_i(t) - \text{EG}(t) - \text{sgn}(\text{SOC}(t) - 20\%)\mathcal{B}_{\text{power}}\} \tag{147}$$

5.5 Numeric Simulation

In this section, we will use numeric simulations to verify our feasibility analysis of real-time energy management in the micro-grid.

5.5.1 Simulation Setup

First, we consider a micro-grid with the on-site renewable energy and fossil fuel generation. We assume that the renewable energy has the generation as shown in Figure 14. Moreover, we assume that the fossil fuel can output the constant power of 100kW.

Then, we consider four types of electrical loads in the micro-grid. The first type of electrical loads has simple characteristics as follows

$$\begin{aligned}
&C_1(t) = 0.5 \;\; E_1(t) = 80 \quad D_1(t) = 2 \;\; T_1(t) = 2 \;\; F_1(t) = 0 \;\; P_1(t) = 1 \\
&C_2(t) = 0.5 \;\; E_2(t) = 120 \;\; D_2(t) = 2 \;\; T_2(t) = 3 \;\; F_2(t) = 1 \;\; P_2(t) = 2 \\
&C_3(t) = 1.0 \;\; E_3(t) = 160 \;\; D_3(t) = 4 \;\; T_3(t) = 4 \;\; F_3(t) = 1 \;\; P_3(t) = 3
\end{aligned} \tag{148}$$

The second type of electrical loads has multiple internal operation phases and can be

represented as

$$\begin{aligned} &C_4(t)=[0.5,1] \quad E_4(t)=[150,160] \quad D_4(t)=4 \;\; T_4(t)=4 \;\; F_4(t)=[0,0] \quad P_4(t)=4 \\ &C_5(t)=[1,0.5,1] \;\; E_5(t)=[120,80,180] \;\; D_5(t)=4 \;\; T_5(t)=5 \;\; F_5(t)=[0,0,0] \;\; P_5(t)=5 \end{aligned} \tag{149}$$

The third type of electrical loads have precedence constraints and they formulate an comprehensive electrical loads that can be represented as

$$\begin{aligned} C_5(t)=[0.5,1,0.5,1.5,0.5] \quad E_5(t)=[50,120,30,140,80] \\ D_5(t)=6 \;\; T_5(t)=6 \;\; F_5(t)=[0,1,0,1,0] \;\; P_5(t)=[6,7,8,9,10] \end{aligned} \tag{150}$$

The last type of electrical load is an AC operating dynamically according to the outside temperature TP_{out}. According to Equation (113), we have this electrical load represented

$$C_6(t) = T_6(t)\, u \;\; E_6(t) = 120 \;\; D_6(t) = 2 \;\; T_6(t) = 2 \;\; F_6(t) = 0 \;\; P_6(t) = 8 \tag{151}$$

where u is the duty cycle as $u = \frac{1}{400}(70 - TP_{\text{out}})$.

Finally, we assume the battery bank has the nominal voltage of 400V and the capacity of 450Ah. The C-rate of the battery bank is 0.5. According to Remark 5.2.7 and Remark 5.2.8, we have the value of $\mathcal{B}_{\text{capacity}}$ and $\mathcal{B}_{\text{power}}$ as

$$\mathcal{B}_{\text{capacity}} = 180 \text{ kWh} \quad \mathcal{B}_{\text{power}} = 90 \text{ kW} \tag{152}$$

Figure 16 shows the power demand of the above electrical loads in the micro-grid under the real-time energy management. The solid blue line represents the total on-site energy generation including the wind power and fossil fuel. Since the fossil fuel outputs the constant power of 100 kW, the total on-site generation in Figure 16 has the same pattern as the wind power in Figure 14. The area with cross line represents the power demand of all electrical loads within 24 hours. As it shows, the power demand exceeds the on-site generation at some time points. At these time points, if

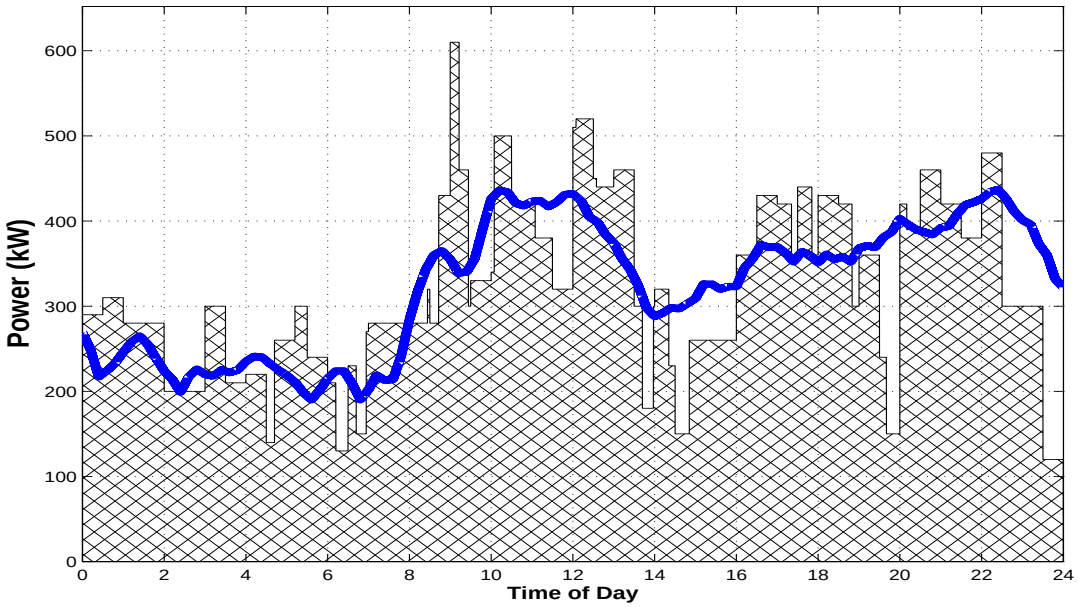


Figure 16: Power Demand under Real-time Energy Management

the battery can provide enough energy to cover the extra power demand, the micro-grid will still be able to run independently. Otherwise, the micro-grid cannot run independently.

5.5.2 Feasibility Verification

In this section, we will show that our feasibility analysis can accurately predict when and how much power is insufficient for the independent operation of the micro-grids.

As shown in Figure 16, the power demand exceeds the on-site generation at some time points. To guarantee the independent operation of the micro-grid, the battery storage will provide some energy to electrical loads. Figure 17 shows the battery dynamics within 24 hours. The green line represents the SOC of the battery. The evolution of the battery SOC is derived as part of the dynamic timing model in Section 5.2.2. The battery SOC increases when the battery is charging and decreases when the battery is discharging. For example, we can see from Figure 16 that the battery is charging within the time interval $[14.4, 16]$ because the on-site generation is larger than the total power demand. This observation exactly matches the increase

of the battery SOC within $[14.4, 16]$ as shown in Figure 17. Moreover, the solid red area represents the discharging power of the battery. The faster the battery SOC decreases, the larger the discharging power is.

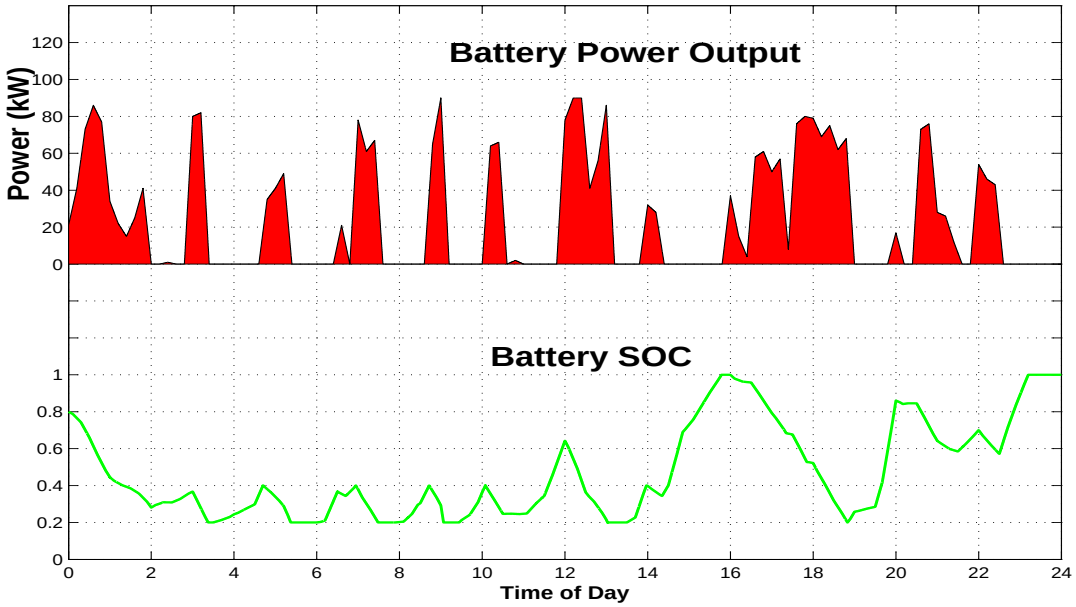


Figure 17: Battery output and SOC

However, even with the supply from the battery storage, the micro-grid may still not be able to run independently. Figure 18 shows when and how much power is insufficient for the independent operation of micro-grids. It is derived from our feasibility analysis studied in Section 5.4.

To verify the result of the feasibility analysis, we mark the results of Figure 17 and Figure 18 using different colors, as illustrated in Figure 19. The red color denotes the discharging power of the battery, which is same as that in Figure 17. The grey color denotes the amount of insufficient power for the independent operation of the micro-grids, which is same as that in Figure 18. As we can see, the combination of the red area and grey area exactly cover the extra power demand exceeding the on-site energy generation (solid blue line). Therefore, our feasibility analysis can accurately predict when and how much power is insufficient for the independent operation of the micro-grid.

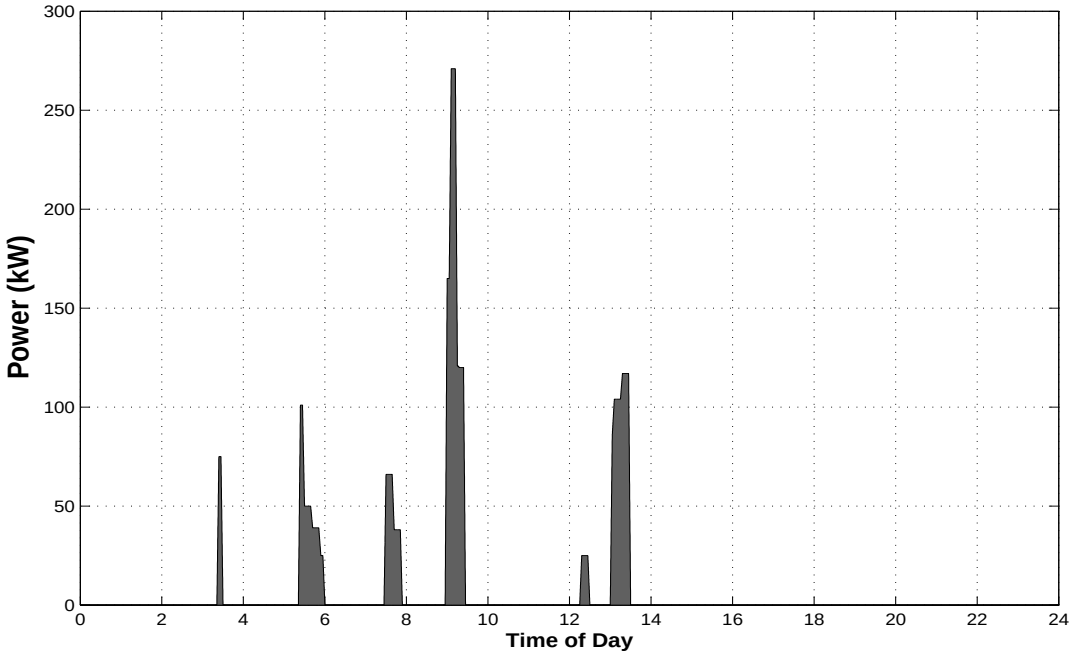


Figure 18: Power Deficiency

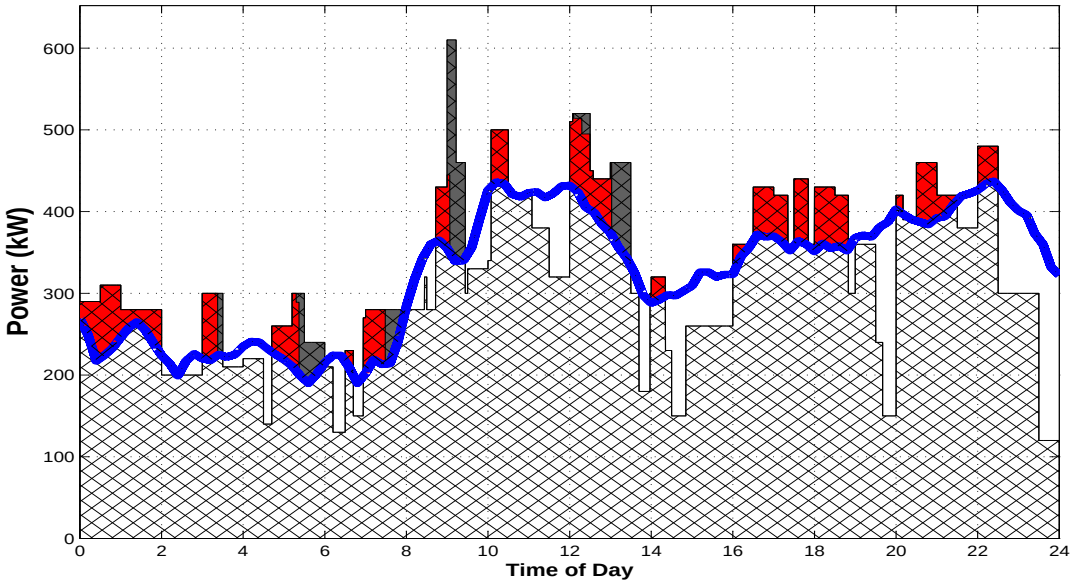


Figure 19: Power Demand under Real-time Energy Management

CHAPTER VI

CONCLUSION AND FUTURE RESEARCH

6.1 Conclusion

The main contributions of this thesis include scheduling analysis for three types of real-time systems and improved system design based on our scheduling analysis. We introduce a novel mathematical timing model that describes dynamic behaviors of multiple operations scheduled in different types of real-time systems. Based on this timing model, we derive mathematical expression for non-worst-case response time of each operation. The timing model and non-worst-case response time analysis are both verified by comparing our mathematical results with the results generated through Truetime Simulation. Using the non-worst-case response time improves systems design for each type of real-time systems studied in this dissertation. For real-time computing system, we derive an necessary and sufficient dynamic schedualbility test, and a measure of robustness. For real-time communication network, we design a model predictive controller based on the controller area network with desirable performance. For real-time energy management, we derive an feasibility analysis for renewable energy based micro-grids. The detailed contribution of this thesis is summarized as follows.

- *Analytical timing model of different types of real-time systems.* The analytical timing model is a set of equations that describes the evolution of operations over time. It reveals internal dynamics of scheduling operations in different types of real-time systems. Each analytical timing model fully consider the characteristics of the corresponding real-time systems. Based on these analytical timing models, we calculate the non-worst-case response time of operations.

- *Dynamic schedulability test of real-time computing systems.* The dynamic schedulability test is a least conservative schedualbility test because it is necessary and sufficient. The goal of the dynamic schedulability test is to check whether a set of operations can meet their deadlines within a finite time interval. The finite time window will slide forward as time propagates. The feature makes the dynamic schedulability test an online test as it consistently takes account for the perturbations happening within the finite time interval. We generalize the dynamic schedulability test to both periodic and non-periodic tasks.

- *Robustness measure of real-time computing systems.* We define a robustness measure as the maximum strength of perturbations on the computing times of scheduled tasks that will not cause loss of schedulability. The robustness measures are able to account for situations at runtime that are unexpected at the design stage. We theoretically justify that tasks are schedulable if the perturbation is smaller than the robustness of the system. Also, we verify that the real-time system with a higher measure of robustness measure is more tolerable with perturbations than the real-time system with a smaller measure of robustness.

- *Effective Model Predictive Control based on the CAN bus.* Model predictive control designs rely on accurate, high fidelity process models. However, real-time scheduling of messages on the CAN bus incurs time-varying delays into feedback control loops. Such time-varying delays make it difficult to derive an reliable process model for MPC design.We propose a MPC design approach that can effectively address time-varying delays in feedback control loop. As a first step, we establish a hybrid timing model of the CAN bus. The hybrid timing model accurately represents the real-time scheduling of message on the CAN bus through a non-block, deterministic hybrid automaton. This hybrid timing

model can continuously adapt to messages changing at runtime. After that, we integrate this hybrid timing model as timing constraints for MPC design. Simulation shows that this MPC design approach based on the hybrid timing model can lead to improved MPC performance.

- *Feasibility Analysis of Micro-grids.* We establish mathematical expressions for electrical loads in the micro-grid by considering their characteristics and constraints. Base on these mathematical models, we develop an analytical timing model that describes the dynamic behavior of scheduling electrical loads under real-time energy management. Our feasibility analysis can predict whether the micro-grid in island mode can generate enough power to support the execution of electrical loads. In case the power is insufficient to supply load demands, we can have accurate information about the amount of insufficient power and the time when the insufficiency happens.

6.2 Future Research

In this section, we will discuss the future direction of the research presented in this thesis.

6.2.1 Multiprocessor Schedulability Test

Our dynamic schedulability test is developed for multiple tasks running on the single processor. In recent years, a strong trend for using multiprocessor on embedded devices has been observed in automotive electronics, and space/satellite systems where the European Space Agency (ESA) is supporting development of suitable multicore processors (LEON3). Mobile phones and related devices already exploit multiprocessor platforms. Therefore, extending our dynamic schedulability test to multiprocessor platform will meet the need of the technology development.

Multiprocessor scheduling can be generally classified into three categories according to the allocation of tasks.

1. No migration. Each task is allocated to a processor and no migration is permitted;

2. Task-level migration. The jobs of a task may execute on different processors; however, each job can only execute on a single processor.

3. Job-level migration. A single job can migrate to and execute on different processors; however, parallel execution of a job is not permitted.

For the first case, our dynamic schedulaiblity test can be directly applied as each task is determined to execute on the same processor. The future research needs to focus on the extension of our dynamic schedulability test on the second and third case.

6.2.2 Time-triggered and Event-triggered Network

Our research focuses on the design of control system based on the CAN bus, which is an event-triggered communication network. In recent years, many time-triggered real-time communication network have emerged. Time-triggered communication means that activities are triggered by the elapsing of time segments. In a time-triggered communication system, message transmission are predetermined at the design stage of a system.

TTCAN (time-triggered CAN) protocol is an extension to the existing CAN protocol. It provides mechanisms to schedule CAN messages in a time-triggered way as well as in an event-triggered way. In vehicles data traffic must usually be both event-triggered (e.g. a temperature change in the cooling mechanism) and time-triggered (e.g. the status of the brakes). Using TTCAN increases the real-time performance in vehicle networks.

FlexRay is another type of real-time communication network designed for both event-triggered and time-triggered transmission. Depending on the application needs, the FlexRay communication cycle is divided into static and dynamic segments. The static data transmission enables time triggered communication to meet the requirement of dependable systems. The dynamic transmission allows each node to use the full bandwidth for event driven communications.

The future direction of this part of research is to derive the hybrid timing model for both event-triggered and time-triggered communication. Such work will extend the application of hybrid timing model to more areas.

6.2.3 Cost Analysis of Micro-grid

Feasibility analysis is a fundamental step that guarantees the normal operation of the micro-grid under appropriate energy management. One of the most important motivation behind the micro-grids is the economic benefit. The future research direction of energy management can focus on optimizing the operation/implementation cost of the micro-grid, under the feasibility constraint.

Another future direction can be the feasibility analysis of a cluster of micro-grids. Each micro-grid is not running independently but connected with other micro-grids. Such connection among micro-grids will improve the robustness and the stability of the whole system.

REFERENCES

[1] Abdallah, A., Wolf, W., and Hellestrand, G., "Statistical Characterization of Execution Time Through Simulation," in *Proc. of International Workshop on Intelligent Solutions in Embedded Systems*, pp. 1–13, 2008.

[2] Abdelzaher, T. F., Sharma, V., and Lu, C., "A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 334–350, 2004.

[3] Abdelzaher, T. and Lu, C., "Schedulability analysis and utilization bounds for highly scalable real-time services," *Proc. of IEEE 7th Real-Time Technology and Applications Symposium*, pp. 15–25, 2001.

[4] Albadi, M. and El-Saadany, E., "A summary of demand response in electricity markets," *Electric Power Systems Research*, vol. 78, pp. 1989–1996, Nov. 2008.

[5] Andersson, B. and Ekelin, C., "Exact Admission-Control for Integrated Aperiodic and Periodic Tasks," in *Proc. of IEEE 11th Symposium on Real-Time and Embedded Technology and Applications*, pp. 76–85, 2005.

[6] Anta, A. and Tabuada, P., "On the Benefits of Relaxing the Periodicity Assumption for Networked Control Systems over CAN," in *Proc. of IEEE International Conference on Real-Time Systems Symposium*, pp. 3–12, 2009.

[7] ÅRzén, K.-E., Cervin, A., Eker, J., and Sha, L., "An Introduction to Control and Scheduling Co-design," in *Proc. of IEEE Conference on Decision and Control*, pp. 4865–4870, 2000.

[8] Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J., "Hard real-time scheduling: The deadline monotonic approach," in *Proc. of IEEE Workshop on Real-Time Operating Systems and Software*, pp. 127–132, 1991.

[9] Axer, P., Sebastian, M., and Ernst, R., "Probabilistic response time bound for CAN messages with arbitrary deadlines," in *Proc. of Design, Automation and Test in Europe Conference and Exhibition*, pp. 1114–1117, Mar. 2012.

[10] Bagherian, A. and Tafreshi, S. M., "A developed Energy Management System for a Microgrid in the Competitive Electricity Market," in *Proc. of 2009 IEEE Bucharest Power Tech Conference*, pp. 1–6, June 2009.

[11] Baruah, S. K., Burns, A., and Davis, R., "Response-Time Analysis for Mixed Criticality Systems," in *Proc. of IEEE 32nd Real-Time Systems Symposium*, pp. 34–43, 2011.

[12] Baruah, S. K., Mok, A. K., and Rosier, L. E., "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," in *Proc. of IEEE Real-Time Systems Symposium*, pp. 182–190, 1990.

[13] Bini, E., Buttazzo, G., and Buttazzo, G., "Rate Monotonic Analysis: the Hyperbolic Bound," *IEEE Transactions on Computers*, vol. 52, pp. 933–942, July 2003.

[14] Boyd, S. and Vandenberghe, L., *Convex Optimization*. Cambridge University Press, 2004.

[15] Bril, R. J., Verhaegh, W. F., and Pol, E.-J. D., "Initial Values for On-line Response Time Calculations," in *Proc. of 15th Euromicro Conference on Real-Time Systems*, pp. 13–22, 2003.

[16] Broster, I., Burns, A., and Rodríguez-Navas, G., "Timing Analysis of Real-Time Communication under Electromagnetic Interference," *Real-Time Systems*, vol. 30, pp. 55–81, 2005.

[17] Broster, I. and Burns, A., "Probabilistic analysis of CAN with faults," in *Proc. of IEEE 23rd Real-Time Systems Symposium*, pp. 269–278, 2002.

[18] Buttazzo, G. C., Lipari, G., Caccamo, M., and Abeni, L., "Elastic Scheduling for Flexible Workload Management," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 289–302, 2002.

[19] Casparsson, L., Rajnak, A., Tindell, K., and Malmberg, P., "Volcano-a revolution in on-board communications," tech. rep., Volvo Technology Report, 1998.

[20] Cervin, A., Henriksson, D., Lincoln, B., Eker, J., and Å rzén, K.-E., "How Does Control Timing Affect Performance," *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, 2003.

[21] Davis, R. I., Burns, A., Bril, R. J., and Lukkien, J. J., "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[22] Davis, R. I., Kollmann, S., Pollex, V., and Slomka, F., "Controller Area Network (CAN) Schedulability Analysis with FIFO queues," in *Proc. of Euromicro Conference on Real-Time Systems*, pp. 45–56, 2011.

[23] Di Natale, M., Zeng, H., Giusto, P., and Ghosal, A., *Understanding and Using the Controller Area Network Communication Protocol*. New York, NY: Springer, 2012.

[24] Facchinetti, T., Bini, E., and Bertogna, M., "Reducing the Peak Power through Real-Time Scheduling Techniques in Cyber-Physical Energy Systems," in *Proc. of 1st International Workshop on Energy Aware Design and Analysis of Cyber Physical Systems*, pp. 1–8, 2010.

[25] Facchinetti, T. and Vedova, M. L. D., "Real-Time Modeling for Direct Load Control in Cyber-Physical Power Systems," *IEEE Transactions on Industrial Electronics*, vol. 7, no. 4, pp. 689–698, 2011.

[26] Farhangi, H., "The Path of the Smart Grid," *IEEE Power and Energy Magazine*, vol. 8, no. 1, pp. 18–28, 2010.

[27] Fisher, N. and Baruah, S., "A Fully Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Arbitrary Relative Deadlines," in *Proc. of 17th Euromicro Conference on Real-Time Systems*, pp. 117–126, 2005.

[28] Gmbh, R. B., "CAN Specification Version 2.0," tech. rep., Stuttgart, Germany, 1991.

[29] Goodwin, G. C., Haimovich, H., Quevedo, D. E., and Welsh, J. S., "A Moving Horizon Approach to Networked Control System Design," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1427–1445, 2004.

[30] Goswami, D., Masrur, A., Schneider, R., Xue, C. J., and Chakraborty, S., "Multirate Controller Design for Resource- and Schedule-Constrained Automotive ECUs," in *Proc. of 16th Conference for Design, Automation and Test in Europe*, no. ii, pp. 1–4, 2013.

[31] Grenier, M., Havet, L., and Navet, N., "Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost," in *Proc. of European Congress Embedded Real Time Software*, 2008.

[32] Guan, N., Ekberg, P., Stigge, M., and Yi, W., "Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems," in *Proc. of IEEE 32nd Real-Time Systems Symposium*, pp. 13–23, Nov. 2011.

[33] Harbour, M. G., Klein, M. H., and Lehoczky, J. P., "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority," in *Proc. of IEEE 12th Real-Time Systems Symposium*, pp. 116–128, 1991.

[34] Hatziargyriou, N. D. and Meliopoulos, A. P. S., "Distributed Energy Sources: Technical Challenges," in *Proc. of IEEE Power Engineering Society Winter Meeting*, vol. 00, pp. 1017–1022, 2002.

[35] Hayden, E., "INTRODUCTION TO MICROGRIDS," *SECURICON Report*, pp. 1–13, 2013.

[36] Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., and Ernst, R., "System Level Performance Analysis - the SymTA / S Approach," *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[37] Imer, O. C., Yüksel, S., and Baar, T., "Optimal Control of LTI Systems over Unreliable Communication Links," *Automatica*, vol. 42, pp. 1429–1439, Sept. 2006.

[38] Jeon, J. M., Kim, D. W., Kim, H. S., Cho, Y. J., and Lee, B. H., "An Analysis of Network-Based Control System using CAN (Controller Area Network) Protocol," in *Proc. of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3577–3581, 2001.

[39] Jiayi, H., Chuanwen, J., and Rong, X., "A review on distributed energy resources and MicroGrid," *Renewable and Sustainable Energy Reviews*, vol. 12, pp. 2472–2483, Dec. 2008.

[40] Joseph, M. and Pandya, P., "Finding response times in a real-time system," *BCS Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[41] Kang, J., Duncan, S. J., and Mavris, D. N., "Real-time Scheduling Techniques for Electric Vehicle Charging in Support of Frequency Regulation," *Procedia Computer Science*, vol. 16, pp. 767–775, Jan. 2013.

[42] Katiraei, F., Iravani, R., Hatziargyriou, N., and Dimeas, A., "Microgrids Management," *IEEE Power and Energy Magazine*, vol. 6, no. 3, pp. 54–65, 2008.

[43] Khan, D. A. and Bril, R. J., "Integrating hardware limitations in CAN schedulability analysis," in *Proc. of IEEE International Workshop on Factory Communication Systems*, pp. 207–210, 2010.

[44] Khan, D. A., Davis, R. I., and Navet, N., "Schedulability analysis of CAN with Non-abortable Transmission requests," in *Proc. of IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1–8, Sept. 2011.

[45] Kroposki, B., Lasseter, R., Ise, T., and Morozumi, S. Papatlianassiou, S. Hatziargyriou, N., "Making Microgrids Work," *IEEE Power and Energy Magazine*, vol. 6, no. 3, pp. 41–53, 2008.

[46] Lehoczky, J., Sha, L., and Ding, Y., "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proc. of IEEE 10th Real-Time Systems Symposium*, pp. 166–171, 1989.

[47] Lehoczky, J. P., "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *Proc. of IEEE 11th Real-time Systems Symposium*, pp. 201–209, 1990.

[48] Li, Z., Huang, P.-C., Mok, A. K., Nghiem, T., Behl, M., Pappas, G., and Mangharam, R., "On the Feasibility of Linear Discrete-Time Systems of the Green Scheduling Problem," in *Proc. of IEEE 32nd Real-Time Systems Symposium*, pp. 295–304, Nov. 2011.

[49] Liu, C. L. and Layland, J. W., "Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment Scheduling Algorithms for Multiprogramming," *Journal of the Association for Computing Machinery*, vol. 20, pp. 46–61, 1973.

[50] Liu, G.-P., Xia, Y., Chen, J., Rees, D., and Hu, W., "Networked Predictive Control of Systems with Random Network Delays in Both Forward and Feedback Channels," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 3, pp. 1282–1297, 2007.

[51] Loontang, P. and de Silva, C. W., "Compensation for Transmission Delays in an Ethernet-Based Control Network Using Variable-Horizon Predictive Control," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 4, pp. 707–718, 2006.

[52] Lu, W.-C., Hsieh, J.-W., Shih, W.-K., and Kuo, T.-W., "A faster exact schedulability analysis for fixed-priority scheduling," *Journal of Systems and Software*, vol. 79, pp. 1744–1753, Dec. 2006.

[53] Lu, Y., Nolte, T., Kraft, J., and Norstrom, C., "A Statistical Approach to Response-Time Analysis of Complex Embedded Real-Time Systems," in *Proc. of IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 153–160, 2010.

[54] Marnay, C., Venkataramanan, G., Stadler, M., a.S. Siddiqui, Firestone, R., and Chandran, B., "Optimal Technology Selection and Operation of Commercial-Building Microgrids," *IEEE Transactions on Power Systems*, vol. 23, pp. 975–982, Aug. 2008.

[55] Martí, P., Camacho, A., Velasco, M., and Gaid, M. E. M. B., "Runtime Allocation of Optional Control Jobs to a Set of CAN-Based Networked Control Systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 503–520, 2010.

[56] Martí, P., Fuertes, J. M., Fohler, G., and Ramamritham, H., "Jitter Compensation for Real-Time Control Systems," in *Proc. of IEEE International Conference on Real-Time Systems Symposium*, pp. 39–48, 2001.

[57] Meliopoulos, A. P., Polymeneas, E., Tan, Z., Huang, R., and Zhao, D., "Advanced Distribution Management System," *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 2109 – 2117, 2013.

[58] Meliopoulos, A. P., Cokkinides, G., Huang, R., Farantatos, E., Choi, S., Lee, Y., and Yu, X., "Smart Grid Technologies for Autonomous Operation and Control," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 1–10, 2011.

[59] Mohamed, F. A. and Koivo, H. N., "System modelling and online optimal management of MicroGrid using Mesh Adaptive Direct Search," *International Journal of Electrical Power and Energy Systems*, vol. 32, pp. 398–407, June 2010.

[60] Mok, A. K.-L., *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, MIT, 1983.

[61] Montestruque, L. A. and Antsaklis, P., "Stability of Model-Based Networked Control Systems With Time-Varying Transmission Times," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1562–1572, Sept. 2004.

[62] Morais, H., Kádár, P., Faria, P., Vale, Z. A., and Khodr, H., "Optimal Scheduling of a Renewable Micro-grid in an Isolated Load Area using Mixed-integer Linear Programming," *Renewable Energy*, vol. 35, pp. 151–156, Jan. 2010.

[63] Natale, M. D., "Evaluating message transmission times in Controller Area Networks without buffer preemption," in *Proc. of Brazilian Workshop on Real-Time Systems*, 2006.

[64] Nghiem, T., Behl, M., Pappas, G. J., and Mangharam, R., "Green Scheduling : Scheduling of Control Systems for Peak Power Reduction," in *Proc. of 2nd International Green Computing Conference and Workshops*, pp. 1–8, 2011.

[65] Nghiem, T. X., Behl, M., Mangharam, R., and Pappas, G. J., "Green scheduling of control systems for peak demand reduction," in *Proc. of IEEE Conference on Decision and Control*, pp. 5131–5136, Dec. 2011.

[66] Palencia, J. C. and Harbour, M. G., "Schedulability Analysis for Tasks with Static and Dynamic offsets," in *Proc. of IEEE 19th Real-Time Systems Symposium*, pp. 26–37, 1998.

[67] Pazul, K., "Controller Area Network (CAN) Basics," tech. rep., Microchip Technology Inc., 1999.

[68] Petters, S. M., Lawitzky, M., Heffernan, R., and Elphinstone, K., "Towards Real Multi-criticality Scheduling," in *Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 155–164, Aug. 2009.

[69] Punnekkat, S., Hansson, H., and Norstrom, C., "Response time analysis under errors for CAN," in *Proc. of IEEE Real-Time Technology and Applications Symposium*, pp. 258–265, 2000.

[70] Rajkumar, R., Sha, L., Lehoczky, J., and Ramamritham, K., "An Optimal Priority Inheritance Policy for Synchronization in Real-Time Systems," in *Advances in Real-time Systems*, pp. 249 – 271, Prentice-Hall, 1995.

[71] Rawlings, J. B., "Tutorial Overview of Model Predictive Control," *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[72] Ribeiro, L. A. d. S., Saavedra, O. R., de Lima, S. L., and Matos, J. G. D., "Isolated Micro-Grids with Renewable Hybrid Generation: The Case of Lencois Island," *IEEE Transactions on Sustainable Energy*, vol. 2, no. 1, pp. 1–11, 2011.

[73] Roe, C., Meliopoulos, S., Entriken, R., and Chhaya, S., "Simulated Demand Response of a Residential Energy Management System," in *Proc. of 2011 IEEE EnergyTech*, pp. 1–6, 2011.

[74] Sha, L., Abdelzaher, T., Å rzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., and Mok, A. K., "Real Time Scheduling Theory: A Historical Perspective," *Real-Time Systems*, vol. 28, pp. 101–155, 2004.

[75] Sha, L., Rajkumar, R., and Lehoczky, J. P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[76] Sjödin, M. and Hansson, H., "Improved Response-Time Analysis Calculations," in *Proc. of IEEE 19th Real-Time Systems Symposium*, pp. 399–408, 1998.

[77] Subramanian, A., Garcia, M., Dominguez-Garcia, A., Callaway, D., Polla, K., and Varaiya, P., "Real-time Scheduling of Deferrable Electric Loads," in *Proc. of American Control Conference*, pp. 3643–3650, 2012.

[78] Tabuada, P., Pappas, G. J., and Lima, P., "Compositional Abstractions of Hybrid Control Systems," *Discrete Event Dynamic Systems*, vol. 14, pp. 203–238, Apr. 2004.

[79] Tindell, K., Burns, A., and Wellings, A., "Calculating Controller Area Network (CAN) Message Response Times," *Control Engineering Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[80] Tindell, K. W., Hansson, H., and Wellings, A. J., "Analysing real-time communications: controller area network (CAN)," in *Proc. of IEEE International Conference on Real-Time Systems Symposium*, pp. 259–263, Dec. 1994.

[81] Tindell, K., "Adding time-offsets to schedulability analysis," tech. rep., University of York, 1994.

[82] Tindell, K. and Burns, A., "Guarantee Message Latency on Control Area Network(CAN)," in *Proc. of International CAN Conference*, pp. 1–11, 1994.

[83] Tindell, K., "An Extendible Approach for Analysing Fixed Priority Hard Real-time Tasks," *Real-Time Systems*, vol. 6, no. 2, pp. 133–151, 1994.

[84] Tomlin, C. J., Mitchell, I., and Bayen, A., "Verification of Hybrid Systems," in *Encyclopedia of Life Support Systems*, 2005.

[85] Tsikalakis, A. G. and Hatziargyriou, N. D., "Centralized Control for Optimizing Microgrids Operation," *IEEE Transactions on Energy Conversion*, vol. 23, no. 1, pp. 241–248, 2008.

[86] Wang, L., *Model Predictive Control System Design and Implementation Using MATLAB*. Springer, 2009.

[87] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., and Stenstr, P., "The Worst-Case Execution Time Problem: Overview of Methods and Survey of Tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–47, 2008.

[88] Xia, F. and Sun, Y., *Control and Scheduling Codesign*. Springer, 2008.

[89] Xu, J. and Parnas, D., "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Transactions on Software Engineering*, vol. 16, pp. 360–369, Mar. 1990.

[90] Xu, Y. and Pan, F., "Scheduling for charging plug-in hybrid electric vehicles," in *Proc. of 51st IEEE Conference on Decision and Control*, pp. 2495–2501, Dec. 2012.

[91] Yomsi, P. M., Bertrand, D., Navet, N., and Davis, R. I., "Controller Area Network (CAN): Response Time Analysis with Offsets," in *Proc. of IEEE International Workshop on Factory Communication Systems*, pp. 43–52, 2012.

[92] Zeng, H., Natale, M. D., Giusto, P., and Sangiovanni-Vincentelli, A., "Stochastic Analysis of CAN-Based Real-Time Automotive Systems," *IEEE Transactions on Industrial Informatics*, vol. 5, pp. 388–401, Nov. 2009.

[93] Zeng, H., Natale, M. D., Giusto, P., and Sangiovanni-vincentelli, A., "Using Statistical Methods to Compute the Probability Distribution of Message Response Time in Controller Area Network," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 678–691, 2010.

[94] Zhang, D. and Papageorgiou, L. G., "Optimal Scheduling of Smart Homes Energy Consumption with Microgrid," in *Proc. of 1st International Conference on Smart Grid, Green Communications and IT Energy-aware Technologies*, pp. 70–75, 2011.

[95] Zhang, F. and Burns, A., "Schedulability Analysis for Real-Time Systems with EDF Scheduling," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.

[96] Zhang, F., Shi, Z., and Mukhopadhyay, S., "Robustness analysis for battery-supported cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, pp. 1–27, Mar. 2013.

[97] Zhang, F., Szwaykowska, K., Wolf, W., and Mooney, V., "Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems," in *Proc. of IEEE Real-Time Systems Symposium*, pp. 47–56, Nov. 2008.